united nations
educational, scientific
and cultural
organization

international atomic
energy agency

the
abdus salam
international centre for theoretical physics

SMR 1331/13

AUTUMN COLLEGE ON PLASMA PHYSICS

8 October - 2 November 2001

COMPUTATIONAL PHYSICS

These are preliminary lecture notes, intended only for distribution to participants.

strada costiera, 11 - 34014 trieste italy - tel. +39 0402240111 fax +39 040224163 - sci_info@ictp.trieste.it - www.ictp.trieste.it

Computational Physics
Ordinary Differential Equations

1.1 Types of Differential Equation

In this chapter we will consider the methods of solution of the sorts of ordinary differential equations (ODEs) which occur very commonly in physics. By ODEs we mean equations involving derivatives with respect to a single variable, usually time. Although we will formulate the discussion in terms of linear ODEs for which we know the analytical solution, this is simply to enable us to make comparisons between the numerical and analytical solutions and does not imply any restriction on the sorts of problems to which the methods can be applied. In the practical work you will encounter examples which do not fit neatly into these categories.

The work in this section is also considered in chapter II of Potter (1973) and chapter 15 of Press et al. (1989).

We consider 3 basic differential equations:

$$\frac{\mathrm{d}y}{\mathrm{d}t} + \alpha y = 0 \quad \Rightarrow \quad y = y_0 \exp(-\alpha t) \qquad \text{The } \textit{Decay} \text{ equation} \tag{1.1a}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} - \alpha y = 0 \quad \Rightarrow \quad y = y_0 \exp(+\alpha t) \qquad \text{The } \textit{Growth} \text{ equation} \tag{1.1b}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} \pm \mathrm{i}\omega y = 0 \quad \Rightarrow \quad y = y_0 \exp(\mp \mathrm{i}\omega t) \qquad \text{The } \textit{Oscillation} \text{ equation} \tag{1.1c}$$

which are representative of most more complex cases.

Higher order differential equations can be reduced to 1st order by appropriate choice of additional variables. The simplest such choice is to define new variables to represent all but the highest order derivative. For example, the damped harmonic oscillator equation, usually written as

$$m\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} + \eta\frac{\mathrm{d}y}{\mathrm{d}t} + \kappa y = 0 \tag{1.2}$$

can be rewritten in terms of y and velocity $v = \mathrm{d}y/\mathrm{d}t$ in the form of a pair of 1st order ODEs

$$\frac{\mathrm{d}v}{\mathrm{d}t} + \frac{\eta}{m}v + \frac{\kappa}{m}y = 0 \tag{1.3a}$$

$$\frac{\mathrm{d}y}{\mathrm{d}t} - v = 0 \tag{1.3b}$$

Similarly any nth order differential equation can be reduced to n 1st order equations.

Such systems of ODEs can be written in a very concise notation by defining a vector, $\boldsymbol{y}$ say, whose elements are the unknowns, such as y and v in (1.3). Any ODE in n unknowns can then be written in the general form

$$\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t} + \boldsymbol{f}(\boldsymbol{y}, t) = 0 \tag{1.4}$$

where $\boldsymbol{y}$ and $\boldsymbol{f}$ are n–component vectors.

Remember that there is no significance in the use of the letter t in the above equations. The variable is not necessarily time but could just as easily be space, as in (1.6), or some other physical quantity.

Formally we can write the solution of (1.4) as

$$\boldsymbol{y}(t) = \boldsymbol{y}(t_0) - \int_{t_0}^{t} \boldsymbol{f}(\boldsymbol{y}(t'), t')\,\mathrm{d}t' \tag{1.5}$$

by integrating both sides over the interval $t_0 \rightarrow t$. Although (1.5) is formally correct, in practice it is usually impossible to evaluate the integral on the right–hand–side as it presupposes the solution $\boldsymbol{y}(t)$. We will have to employ an approximation.

All differential equations require boundary conditions. Here we will consider cases in which all the boundary conditions are defined at a particular value of t (e.g. $t = 0$). For higher order equations the boundary conditions may be defined at different values of t. The modes of a violin string at frequency ω obey the equation

$$\frac{\mathrm{d}^2 y}{\mathrm{d}x^2} = -\frac{\omega^2}{c^2} y \tag{1.6}$$

with boundary conditions such that $y = 0$ at both ends of the string. We shall consider such problems in chapter 2.10.3.

1.2 Euler Method

Consider an approximate solution of (1.5) over a small interval $\delta t = t_{n+1} - t_n$ by writing the integral as

$$\int_{t_n}^{t_{n+1}} \boldsymbol{f}(\boldsymbol{y}(t'), t')\,\mathrm{d}t' \approx \delta t \boldsymbol{f}(\boldsymbol{y}(t_n), t_n). \tag{1.7}$$

to obtain

$$\boldsymbol{y}(t_{n+1}) = \boldsymbol{y}(t_n) - \delta t \boldsymbol{f}(\boldsymbol{y}(t_n), t_n).$$

or, in a more concise notation,

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n - \delta t \boldsymbol{f}(\boldsymbol{y}_n, t_n) = \boldsymbol{y}_n - \delta t \boldsymbol{f}_n. \tag{1.8}$$

We can integrate over any larger interval by subdividing the range into sections of width δt and repeating (1.8) for each part.

Equivalently we can consider that we have approximated the derivative with a *forward* difference

$$\left.\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t}\right|_n \approx \frac{\boldsymbol{y}_{n+1} - \boldsymbol{y}_n}{\delta t}. \tag{1.9}$$

We will also come across centred and backward differences,

$$\left.\frac{\mathrm{d}\boldsymbol{y}}{\mathrm{d}t}\right|_n \approx \begin{cases} \dfrac{\boldsymbol{y}_{n+1} - \boldsymbol{y}_{n-1}}{2\delta t} & \text{centred} \\ \dfrac{\boldsymbol{y}_n - \boldsymbol{y}_{n-1}}{\delta t} & \text{backward} \end{cases} \tag{1.10}$$

respectively.

Here we have used a notation which is very common in computational physics, in which we calculate $\boldsymbol{y}_n = \boldsymbol{y}(t_n)$ at discrete values of t given by $t_n = n\delta t$, and $\boldsymbol{f}_n = \boldsymbol{f}(\boldsymbol{y}_n, t_n)$.

In what follows we will drop the vector notation $\boldsymbol{y}$ except when it is important for the discussion.

1.2.1 Order of Accuracy

How accurate is the Euler method? To quantify this we consider a Taylor expansion of $y(t)$ around t_n

$$y_{n+1} = y_n + \delta t \left.\frac{\mathrm{d}y}{\mathrm{d}t}\right|_n + \frac{\delta t^2}{2}\left.\frac{\mathrm{d}^2 y}{\mathrm{d}t^2}\right|_n + \ldots \tag{1.11}$$

and substitute this into (1.8)

$$y_n + \delta t \left.\frac{\mathrm{d}y}{\mathrm{d}t}\right|_n + \frac{\delta t^2}{2}\left.\frac{\mathrm{d}^2 y}{\mathrm{d}t^2}\right|_n + \ldots \approx y_n - \delta t f(y_n, t_n) \tag{1.12a}$$

$$= y_n + \delta t \left.\frac{\mathrm{d}y}{\mathrm{d}t}\right|_n, \tag{1.12b}$$

where we have used (1.4) to obtain the final form. Hence, we see that the term in δt in the expansion has been correctly reproduced by the approximation, but that the higher order terms are wrong. We therefore describe the Euler method as *1st order accurate.*

An approximation to a quantity is nth order accurate if the term in δt^n in the Taylor expansion of the quantity is correctly reproduced. The order of accuracy of a *method* is the order of accuracy with which the unknown is approximated.

Note that the term *accuracy* has a slightly different meaning in this context from that which you might use to describe the results of an experiment. Sometimes the term *order of accuracy* is used to avoid any ambiguity.

The leading order deviation is called the *truncation error.* Thus in (1.12) the truncation error is the term in δt^2.

1.2.2 Stability

The Euler method is 1st order accurate. However there is another important consideration in analysing the method: *stability.* Let us suppose that at some time the actual numerical solution deviates from the true solution of the difference equation (1.8) (N.B. **not** the original differential equation (1.4)) by some small amount δy, due, for example, to the finite accuracy of the computer. Then adding this into (1.8) gives

$$y_{n+1} + \delta y_{n+1} = y_n + \delta y_n - \delta t \left[f(y_n, t_n) + \left.\frac{\partial f}{\partial y}\right|_n \delta y_n \right], \tag{1.13}$$

where the term in brackets, [], is the Taylor expansion of $f(y,t)$ with respect to y. Subtracting (1.8) we obtain a *linear* equation for δy

$$\delta y_{n+1} = \left[1 - \delta t \left.\frac{\partial f}{\partial y}\right|_n \right] \delta y_n, \tag{1.14}$$

which it is convenient to write in the form

$$\delta y_{n+1} = g \delta y_n. \tag{1.15}$$

If g has a *magnitude* greater than one then δy_n will tend to grow with increasing n and may eventually dominate over the required solution. Hence the Euler method is stable only if $|g| \leq 1$ or

$$-1 \leq 1 - \delta t \frac{\partial f}{\partial y} \leq +1. \tag{1.16}$$

As δt is positive by definition the 2nd inequality implies that the derivative must also be positive. The 1st inequality leads to a restriction on δt, namely

$$\delta t \leq 2 / \frac{\partial f}{\partial y}. \tag{1.17}$$

When the derivative is complex more care is required in the calculation of $|g|$. In this case it is easier to look for solutions of the condition $|g|^2 \leq 1$. For the oscillation equation (1.1c) the condition becomes

$$1 + \delta t^2 \omega^2 \leq 1 \tag{1.18}$$

which is impossible to fulfil for real δt and ω. Comparing these result with our 3 types of differential equations (1.1) we find the following stability conditions

Decay	Growth	Oscillation
$\delta t \leq 2/\alpha$	unstable	unstable

The Euler method is *conditionally stable* for the decay equation.

A method is stable if a small deviation from the true solution does not tend to grow as the solution is iterated.

1.2.3 The Growth Equation

Actually, our analysis doesn't make too much sense in the case of the growth equation as the true solution should grow anyway. A more sensible condition would be that the *relative* error in the solution does not grow. This can be achieved by substituting $y_n \epsilon_n$ for δy_n above and looking for the condition that ϵ_n does not grow. We will not treat this case further here but it is, in fact, very important in problems such as *chaos,* in which small changes in the initial conditions lead to solutions which diverge from one another.

1.2.4 Application to Non–Linear Differential Equations

The linear–differential equations in physics can often be solved analytically whereas most non–linear ones can only be solved numerically. It is important therefore to be able to apply the ideas developed here to such cases.

Consider the simple example

$$\frac{\mathrm{d}y}{\mathrm{d}t} + \alpha y^2 = 0. \tag{1.19}$$

In this case $f(y,t) = \alpha y^2$ and $\partial f/\partial y = 2\alpha y$ which can be substituted into (1.17) to give the stability condition

$$\delta t \leq 1/\alpha y, \tag{1.20}$$

which depends on y, unlike the simpler cases. In writing a program to solve such an equation it may therefore be necessary to monitor the value of the solution, y, and adjust δt as necessary to maintain stability.

1.2.5 Application to Vector Equations

A little more care is required when $\boldsymbol{y}$ and $\boldsymbol{f}$ are vectors. In this case $\delta \boldsymbol{y}$ is an arbitrary infinitesimal vector and the derivative $\partial \boldsymbol{f}/\partial \boldsymbol{y}$ is a matrix $\mathbf{F}$ with components

$$F_{ij} = \frac{\partial f_i}{\partial y_j} \tag{1.21}$$

in which f_i and y_j represent the components of $\boldsymbol{f}$ and $\boldsymbol{y}$ respectively. Hence (1.14) takes the form

$$\delta y_i^{(n+1)} = \delta y_i^{(n)} - \delta t \sum_j \left.\frac{\partial f_i}{\partial y_j}\right|_n \delta y_j^{(n)} \tag{1.22a}$$

$$\delta \boldsymbol{y}_{n+1} = [\mathbf{I} - \delta t \mathbf{F}]\, \delta \boldsymbol{y}_n = \mathbf{G} \delta \boldsymbol{y}_n. \tag{1.22b}$$

This leads directly to the stability condition that **all** the eigenvalues of $\mathbf{G}$ must have modulus less than unity (see problem 6).

In general any of the stability conditions derived in this course for scalar equations can be re–expressed in a form suitable for vector equations by applying it to **all** the eigenvalues of an appropriate matrix.

1.3 The Leap–Frog Method

How can we improve on the Euler method? The most obvious way would be to replace the forward difference in (1.9) with a *centred* difference (1.10) to get the formula

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_{n-1} - 2\delta t \boldsymbol{f}(\boldsymbol{y}_n, t). \tag{1.23}$$

If we expand both $\boldsymbol{y}_{n+1}$ and $\boldsymbol{y}_{n-1}$ as in section 1.2.1 (1.23) becomes

$$\begin{aligned} & y_n + \delta t \left.\frac{\mathrm{d}y}{\mathrm{d}t}\right|_n + \left.\frac{\delta t^2}{2}\frac{\mathrm{d}^2y}{\mathrm{d}t^2}\right|_n + \left.\frac{\delta t^3}{6}\frac{\mathrm{d}^3y}{\mathrm{d}t^3}\right|_n + \ldots \\ &= y_n - \delta t \left.\frac{\mathrm{d}y}{\mathrm{d}t}\right|_n + \left.\frac{\delta t^2}{2}\frac{\mathrm{d}^2y}{\mathrm{d}t^2}\right|_n - \left.\frac{\delta t^3}{6}\frac{\mathrm{d}^3y}{\mathrm{d}t^3}\right|_n + \ldots - 2\delta t f_n \qquad (1.24a) \\ &= y_n + \delta t \left.\frac{\mathrm{d}y}{\mathrm{d}t}\right|_n + \left.\frac{\delta t^2}{2}\frac{\mathrm{d}^2y}{\mathrm{d}t^2}\right|_n - \left.\frac{\delta t^3}{6}\frac{\mathrm{d}^3y}{\mathrm{d}t^3}\right|_n + \ldots \qquad (1.24b) \end{aligned}$$

from which all terms up to δt^2 cancel so that the method is clearly 2nd order accurate. Note in passing that using (1.23) 2 consecutive values of y_n are required in order to calculate the next one: y_n and y_{n-1} are required to calculate y_{n+1}. Hence 2 boundary conditions are required, even though (1.23) was derived from a 1st order differential equation. This so–called *leap–frog* method is more accurate than the Euler method, but is it stable? Repeating the same analysis (section 1.2.2) as for the Euler method we again obtain a linear equation for δy_n

$$\delta y_{n+1} = \delta y_{n-1} - 2\delta t \left.\frac{\partial f}{\partial y}\right|_n \delta y_n. \tag{1.25}$$

We analyse this equation by writing $\delta y_n = g\delta y_{n-1}$ and $\delta y_{n+1} = g^2\delta y_{n-1}$ to obtain

$$g^2 = 1 - 2\delta t \left.\frac{\partial f}{\partial y}\right|_n g \tag{1.26}$$

which has the solutions

$$g = \delta t\frac{\partial f}{\partial y} \pm \sqrt{\left(\delta t\frac{\partial f}{\partial y}\right)^2 + 1}. \tag{1.27}$$

The product of the 2 solutions is equal to the constant in the quadratic equation, i.e. -1. Since the 2 solutions are different, one of them always has magnitude > 1. Since for a small random error it is impossible to guarantee that there will be no contribution with $|g| > 1$ this contribution will tend to dominate as the equation is iterated. Hence the method is unstable.

There is an important exception to this instability: when the partial derivative is purely imaginary (but not when it has some general complex value), the quantity under the square root in (1.27) can be negative and both g's have modulus unity. Hence, for the case of oscillation (1.1c) where $\partial f/\partial y = \pm \mathrm{i}\omega$, the algorithm is just stable, as long as

$$\delta t \leq 1/\omega. \tag{1.28}$$

The stability properties of the leap–frog method are summarised below

Decay	Growth	Oscillation
unstable	unstable	$\delta t \leq 1/\omega$

Again the growth equation should be analysed somewhat differently.

1.4 The Runge–Kutta Method

So far we have found one method which is stable for the decay equation and another for the oscillatory equation. Can we combine the advantages of both?

As a possible compromise consider the following two step algorithm (ignoring vectors)

$$y'_{n+1/2} = y_n - \tfrac{1}{2}\delta t f(y_n, t_n) \tag{1.29a}$$

$$y_{n+1} = y_n - \delta t f(y'_{n+1/2}, t_{n+1/2}). \tag{1.29b}$$

In practice the intermediate value $y'_{n+1/2}$ is discarded after each step. We see that this method consists of an Euler (see section 1.2) step followed by a Leap–Frog (see section 1.3) step. This is called the 2nd order *Runge–Kutta* or *two–step* method. It is in fact one of a hierarchy of related methods of different accuracies.

The stability analysis for (1.29) is carried out in the same way as before. Here we simply quote the result

$$\delta y_{n+1} = \left[1 - \delta t \frac{\partial f}{\partial y} + \tfrac{1}{2}\left(\delta t \frac{\partial f}{\partial y}\right)^2\right] \delta y_n. \tag{1.30}$$

In deriving this result it is necessary to assume that the derivatives, $\partial f/\partial y$ are independent of t. This is not usually a problem.

From (1.30) we conclude the stability conditions

Decay	Growth	Oscillation
$\delta t \leq 2/\alpha$	unstable	$1 + \frac{1}{4}(\delta t \omega)^4 \leq 1$

Note that in the oscillatory case the method is strictly speaking unstable but the effect is so small that it can be ignored in most cases, as long as $\omega \delta t < 1$. This method is often used for damped oscillatory equations.

1.5 The Predictor–Corrector Method

This method is very similar to and often confused with the Runge–Kutta (see section 1.4) method. We consider substituting the trapezoidal rule for the estimate of the integral in (1.5) to obtain the equation

$$\boldsymbol{y}_{n+1} = \boldsymbol{y}_n - \tfrac{1}{2}\delta t \left[\boldsymbol{f}(\boldsymbol{y}_{n+1}, t_{n+1}) + \boldsymbol{f}(\boldsymbol{y}_n, t_n)\right]. \tag{1.31}$$

Unfortunately the presence of $\boldsymbol{y}_{n+1}$ on the right hand side makes a direct solution of (1.31) impossible except for special cases (see section 1.6) below. A possible compromise is the following method

$$y'_{n+1} = y_n - \delta t f(y_n, t_n) \tag{1.32a}$$

$$y_{n+1} = y_n - \tfrac{1}{2}\delta t \left[f(y'_{n+1}, t_{n+1} + f(y_n, t_n)\right]. \tag{1.32b}$$

This method consists of a *guess* for y_{n+1} based on the Euler (see section 1.2) method (the *Prediction*) followed by a *correction* using the trapezoidal rule to solve the integral equation (1.5). The accuracy and stability properties are identical to those of the Runge–Kutta method.

1.6 The Intrinsic Method

Returning to the possibility of solving the integral equation (1.5) using the trapezoidal or trapezium rule (1.31), let us consider the case of a linear differential equation, such as our examples (1.1). For the decay equation we have

$$y_{n+1} = y_n - \tfrac{1}{2}\delta t \left[\alpha y_{n+1} + \alpha y_n\right] \tag{1.33}$$

which can be rearranged into an explicit equation for y_{n+1} as a function of y_n

$$y_{n+1} = \left[\frac{1 - \frac{1}{2}\delta t \cdot \alpha}{1 + \frac{1}{2}\delta t \cdot \alpha}\right] y_n. \tag{1.34}$$

This *intrinsic* method is 2nd order accurate as that is the accuracy of the trapezoidal rule for integration. What about the stability? Applying the same methodology as before we find that the crucial quantity, g, is the expression in square brackets, [], in (1.34) which is always < 1 for the decay equation and has modulus unity in the oscillatory case (after substituting $\alpha \mapsto \pm i\omega$). Hence it is stable in both cases. Why is it not used instead of the other methods? Unfortunately only a small group of equations, such as our examples, can be rearranged in this way. For non–linear equations it may be impossible, and even for linear equations when $\boldsymbol{y}$ is a vector, there may be a formal solution which is not useful in practice. It is always possible to solve the resulting non–linear equation iteratively, using (e.g.) *Newton–Raphson* iteration, but this is usually not worthwhile in practice.

In fact the intrinsic method is also stable for the growth equation when it is analysed as discussed earlier (section 1.2.3), so that the method is in fact stable for all 3 classes of equations.

Decay	Growth	Oscillation
stable	stable	stable

1.7 Summary

In these notes we have introduced some methods for solving ordinary differential equations. However, by far the most important lesson to be learned is that to be useful a method must be both *accurate* and *stable*. The latter condition is often the most difficult to fulfil and careful analysis of the equations to be solved may be necessary before choosing an appropriate method to use for finding a numerical solution.

The stability conditions derived above tend to have the form $\delta t \leq \omega^{-1}$ which may be interpreted as δt should be less than the characteristic period of oscillation. This conforms with common sense. In fact we can write down a more general common sense condition: δt should be small compared with the smallest time scale present in the problem.

Finally, many realistic problems don't fall into the neat categories of (1.1). The simplest example is a damped harmonic oscillator. Often it is difficult to find an exact analytical solution for the stability condition. It pays in such cases to consider some extreme conditions, such as (e.g.) very weak damping or very strong damping, work out the conditions for these cases and simply choose the most stringent condition. In non–linear problems the cases when the unknown, y, is very large or very small may provide tractable solutions.

1.8 Problems

1. Write down definitions of the terms *order of accuracy*, *truncation error*, *conditional stability* as applied to the numerical solution of ordinary differential equations.

2. Write down 1st order accurate finite difference approximations for

$$\text{a) } \frac{\mathrm{d}f}{\mathrm{d}x} \qquad \text{b) } \frac{\mathrm{d}^2 f}{\mathrm{d}x^2} \qquad \text{c) } \frac{\mathrm{d}^3 f}{\mathrm{d}x^3} \qquad \text{d) } \frac{\partial}{\partial t}\frac{\partial f}{\partial x}$$

   Hint: the result has to be something like $ay_{n+1} + by_n + cy_{n-1}$. Expand the ys around y_n and choose the coefficients to eliminate contributions from unwanted terms in the expansion.

3. Derive expressions for the truncation error of the following difference approximations.

$$\text{a) } \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_n = \frac{f_{n+1} - f_n}{\delta x} \qquad \text{b) } \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_n = \frac{f_{n+1} - f_{n-1}}{2\delta x}$$

$$\text{c) } \left.\frac{\mathrm{d}^2 f}{\mathrm{d}x^2}\right|_n = \frac{f_{n+1} - 2f_n + f_{n-1}}{\delta x^2} \qquad \text{d) } \left.\frac{\mathrm{d}^3 f}{\mathrm{d}x^3}\right|_n = \frac{f_{n+2} - 2f_{n+1} + 2f_{n-1} - f_{n-2}}{2\delta x^3}$$

$$\text{e) } \left.\frac{\mathrm{d}f}{\mathrm{d}x}\right|_n = \frac{-\frac{1}{12}f_{n+2} + \frac{2}{3}f_{n+1} - \frac{2}{3}f_{n-1} + \frac{1}{12}f_{n-2}}{\delta x}$$

4. The torsion of a bar is described by the differential equation

$$\frac{\mathrm{d}^4\theta}{\mathrm{d}x^4} + \tau\theta = 0.$$

Show how to re–express this as a system of first order differential equations.

5. Write down an expression for solving the differential equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} + \frac{\gamma}{y} = 0$$

by Euler's method and show under what conditions the method is stable.

Write and test a short program (it should only require a few lines) to test the method. Vary δt, γ and $y(t = 0)$ to check the validity of the stability condition you have derived.

6. Using equation 1.22 show that the Euler method is stable for a vector equation provided all the eigenvalues of $\mathbf{G}$ have modulus less than or equal to unity.

7. Show that equation 1.30 gives the correct stability condition for both the Runge–Kutta and Predictor–Corrector methods. Why do you think this is a good method for damped oscillatory equations? (The last part doesn't have to be mathematically rigorous).

1.9 Project — Classical Electrons in a Magnetic Field

The simplest source of ODEs in physics is classical mechanics, so we choose such a problem. The dynamics of a charge in a magnetic field is described by the equation

$$m\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = q\boldsymbol{v} \times \boldsymbol{B} - \gamma\boldsymbol{v} \tag{1.35}$$

where m and q are the mass and charge of the particle respectively, $\boldsymbol{B}$ is the magnetic field and γ represents some sort of friction.

1.9.1 A Uniform Field

Before considering the more general problem we start with a particle in a spatially uniform field. This problem is analytically solvable and can be used to test the various methods before applying them to the general case.

Units

Note firstly that there are only 2 independent constants in the problem, qB/m and γ/m, and that these constants have the units of inverse time; in fact the former is the *cyclotron frequency* and the latter is a *damping rate.* In general in any programming problem it pays to think carefully about the units to be used in the program. There are several reasons for this.

- If inappropriate units are used the program may not work at all. An example of this would be the use of SI units to study the dynamics of galaxies or to study atomic physics. In the former R^3 might easily arise and be bigger than the largest number representable on the machine, whereas in the latter $\hbar^4$ may be smaller than the smallest number on the machine and be set automatically to zero with disastrous consequences.

- The problem often has its own natural units and it makes sense to work in these units. This has the consequence that most of the numbers in your program will be of order unity rather than very large or very small.

In general you should look for the natural units of a problem and write your program appropriately. Note that these will generally not be SI or cgs.

In the problem we are considering here there are 2 *natural* time scales, m/qB and m/γ. If we decide to work in one of these, e.g. the cyclotron period m/qB, we can rewrite (1.35) in the simpler form

$$\frac{\mathrm{d}v_x}{\mathrm{d}t'} = +v_y - \left|\frac{\gamma}{qB}\right| v_x \tag{1.36a}$$

$$\frac{\mathrm{d}v_y}{\mathrm{d}t'} = -v_x - \left|\frac{\gamma}{qB}\right| v_y \tag{1.36b}$$

or perhaps

$$\frac{\mathrm{d}v_x}{\mathrm{d}t'} = -v_y - \left|\frac{\gamma}{qB}\right| v_x \tag{1.37a}$$

$$\frac{\mathrm{d}v_y}{\mathrm{d}t'} = +v_x - \left|\frac{\gamma}{qB}\right| v_y \tag{1.37b}$$

depending on the sign of qB/m. Here $t = t'|m/qB|$ and we have chosen our coordinate system such that the magnetic field, $\boldsymbol{B}$, is in the z–direction. Note, in addition, that choosing the units appropriately has eliminated all but one of the constants from the problem. This cuts down on superfluous arithmetic in the program.

The Analytical Solution

In order to understand the behaviour of the various methods for ODEs we need to know the analytical solution of the problem. 2 dimensional problems such as this one are often most easily solved by turning the 2D vector into a complex number. Thus by defining $\tilde{v} = v_x + \mathrm{i}v_y$ we can rewrite (1.35) in the form

$$\frac{\mathrm{d}\tilde{v}}{\mathrm{d}t} = -\mathrm{i}\tilde{v}\left(\frac{qB}{m}\right) - \tilde{v}\left(\frac{\gamma}{m}\right) \tag{1.38}$$

which can be easily solved using the *integrating factor* method to give

$$\tilde{v} = \tilde{v}_0 \exp\left[-\mathrm{i}\left(\frac{qB}{m}\right)t - \left(\frac{\gamma}{m}\right)t\right]. \tag{1.39}$$

Finally we take real and imaginary parts to find the v_x and v_y components

$$v_x = +v_0 \cos\left[\left(\frac{qB}{m}\right)t + \phi_0\right] \exp\left[-\left(\frac{\gamma}{m}\right)t\right] \tag{1.40a}$$

$$v_y = -v_0 \sin\left[\left(\frac{qB}{m}\right)t + \phi_0\right] \exp\left[-\left(\frac{\gamma}{m}\right)t\right] \tag{1.40b}$$

Choosing an Algorithm

The problem as posed does not fit neatly into the categories defined in (1.1). By considering the accuracy and stability properties of the various algorithms described in these notes you have to decide which is the most appropriate algorithm to use for solving the problem. The computer time required by the algorithm as well as the ease with which it can be programmed may be legitimate considerations. It may not be possible to solve the the stability equations exactly in the most general case. Nevertheless you should be able to deduce enough information on which to base a decision.

In your report you should discuss the merits and demerits of the various possible algorithms and explain the rationale behind your choice. You should also write a program to implement your chosen algorithm and test it for various values of δt and in all the physically significant regimes.

1.9.2 Crossed Electric and Magnetic Fields

You are now in a position to apply your chosen algorithm to a more complicated problem. In addition to the uniform magnetic field, $\boldsymbol{B}$, we now add an electric field in the x–direction, $\boldsymbol{E} = (E_x, 0, 0)$. Thus (1.35a) must be modified to read

$$\frac{\mathrm{d}v_x}{\mathrm{d}t} = +\left(\frac{qB}{m}\right) v_y - \left(\frac{\gamma}{m}\right) v_x + \left(\frac{qE}{m}\right) \tag{1.41a}$$

$$\frac{\mathrm{d}v_y}{\mathrm{d}t} = -\left(\frac{qB}{m}\right) v_x - \left(\frac{\gamma}{m}\right) v_y \tag{1.41b}$$

You should now write a program to solve (1.41) using the most appropriate method as found in section 1.9.1. Try to investigate the behaviour of the system in various physical regimes. You should also vary δt to check whether the stability conforms to your expectations. Think about the physical system you are describing and whether your results are consistent with the behaviour you would expect.

1.9.3 Oscillating Electric Field

Finally, if you have time[1], you might consider making the electric field explicitly time dependent

$$E_x = E_0 \cos \omega t \tag{1.42}$$

and investigate the behaviour of the system as a function of frequency.

1.9.4 Your Report

In your report you should describe how you have chosen the algorithm and the value of δt and show that the method you have chosen is stable for this sort of equation and for the range of parameters you have chosen.

You should think about the physics of the problem and try to identify the physically interesting parameter combinations. In particular you should ask yourself whether there is a qualitative change of behaviour at some value of the parameters or whether there is a value at which some special behaviour might be observed. Let your physical intuition guide your choice of parameters. If you can't identify the physically interesting values, then start by doing a broad sweep of the meaningful parameters to try to identify any interesting features. Once you have found the interesting parameter ranges you can look at them more closely.

Try not to make meaningless changes. If the results can only depend on a particular combination of parameters it is pointless to vary them individually.

Your report should contain a representative selection of results and a discussion of the physics which they illustrate.

Please include a listing of your program as an appendix well as sending it as an e-mail attachment to `Computational-Physics, A. MacKinnon` or `ph.compphys@ic.ac.uk`. Do **not** send these to my personal e-mail address.

[1] There are no marks for this section

Computational Physics
Partial Differential Equations

2.1 Types of Equations

The PDE's which occur in physics are mostly second order[1]. The work in this section is also considered in chapter III of Potter (1973) and chapter 17 of Press et al. (1989).

For linear equations in 2 dimensions there is a simple classification in terms of the general equation

$$a\frac{\partial^2 V}{\partial x^2} + 2b\frac{\partial^2 V}{\partial x \partial y} + c\frac{\partial^2 V}{\partial y^2} + d\frac{\partial V}{\partial x} + e\frac{\partial V}{\partial y} + fV + g = 0 \tag{2.1}$$

as shown in the following table

Condition	Type	Example
$b^2 < ac$	Elliptic	Laplace's equation (2.2a)
$b^2 > ac$	Hyperbolic	Wave equation (2.2b)
$b^2 = ac$	Parabolic	Diffusion/Schrödinger equation (2.2c)

These are listed in their simplest form as follows (with the substitution $y \mapsto t$ where appropriate)

$$\text{Laplace's equation} \qquad \frac{\partial^2 V}{\partial x^2} + \frac{\partial^2 V}{\partial y^2} = 0 \tag{2.2a}$$

$$\text{Wave equation} \qquad \frac{\partial^2 V}{\partial x^2} - \frac{1}{c^2}\frac{\partial^2 V}{\partial t^2} = 0 \tag{2.2b}$$

$$\text{Diffusion equation} \qquad D\frac{\partial^2 V}{\partial x^2} - \frac{\partial V}{\partial t} = 0 \tag{2.2c}$$

We shall consider each of these cases separately as different methods are required for each.

2.2 Elliptic Equations — Laplace's equation

We shall deal with elliptic equations later when we come to consider matrix methods (section 2.10.3). For the moment it suffices to note that, apart from the formal distinction, there is a very practical distinction to be made between elliptic equations on the one hand and hyperbolic and parabolic on the other hand. Generally speaking elliptic equations have boundary conditions which are specified around a closed boundary. Usually **all** the derivatives are with respect to spatial variables, such as in Laplace's or Poisson's Equation.

Hyperbolic and Parabolic equations, by contrast, have at least one open boundary. The boundary conditions for at least one variable, usually time, are specified at one end and the system is integrated indefinitely. Thus, the wave equation and the diffusion equation contain a time variable and there is a set of initial conditions at a particular time.

These properties are, of course, related to the fact that an ellipse is a closed object, whereas hyperbolæ and parabolæ are open.

[1] There are some exceptions such as the 4th order equation which describes torsion of a beam.

2.3 Hyperbolic Equations — Wave equations

The classical example of a hyperbolic equation is the wave equation

$$\frac{\partial^2 y}{\partial t^2} - c^2 \frac{\partial^2 y}{\partial x^2} = 0. \tag{2.3}$$

The wave equation can be rewritten in the form

$$\left(\frac{\partial}{\partial t} + c\frac{\partial}{\partial x}\right)\left(\frac{\partial}{\partial t} - c\frac{\partial}{\partial x}\right) y = 0 \tag{2.4}$$

or as a system of 2 equations

$$\frac{\partial z}{\partial t} + c\frac{\partial z}{\partial x} = 0 \tag{2.5a}$$

$$\frac{\partial y}{\partial t} - c\frac{\partial y}{\partial x} = z. \tag{2.5b}$$

Note that the first of these equations (2.5a) is independent of y and can be solved on it's own. The second equation (2.5b) can then be solved by using the known solution of the first. Note that we could equally have chosen the equations the other way round, with the signs of the velocity c interchanged.

As the 2 equations (2.5) are so similar we expect the stability properties to be the same. We therefore concentrate on (2.5a) which is known as the *Advective* equation and is in fact the conservation of mass equation of an incompressible fluid (section 2.6.1)

$$\frac{\partial u}{\partial t} + c\frac{\partial u}{\partial x} = 0. \tag{2.6}$$

Note also that the boundary conditions will usually be specified in the form

$$u(x,t) = u_0(x) \text{ at } t = 0. \tag{2.7}$$

which gives the value of $u(x,t)$ for all x at a particular time.

2.3.1 A Simple Algorithm

As a first attempt to solve (2.6) we consider using centred differences for the space derivative and Euler's method for the time part

$$u_j^{n+1} = u_j^n - \frac{c\delta t}{2\delta x}\left(u_{j+1}^n - u_{j-1}^n\right) \tag{2.8}$$

where the subscripts $_j$ represent space steps and the superscripts ^{n} time steps. By analogy with the discussion of the Euler (section 1.2) and Leap–Frog (section 1.3) methods we can see immediately that this method is 1st order accurate in t and 2nd order in x. We note firstly that

$$u_j^{n+1} - u_j^n \approx \delta t \left.\frac{\partial u}{\partial t}\right|_j^n + \tfrac{1}{2}\delta t^2 \left.\frac{\partial^2 u}{\partial t^2}\right|_j^n + \ldots \tag{2.9}$$

whereas

$$u_{j+1}^n - u_{j-1}^n \approx 2\delta x \left.\frac{\partial u}{\partial x}\right|_j^n + \tfrac{1}{3}\delta x^3 \left.\frac{\partial^3 u}{\partial x^3}\right|_j^n + \ldots \tag{2.10}$$

and substitute these forms into (2.8) to obtain

$$\delta t \left.\frac{\partial u}{\partial t}\right|_j^n + \tfrac{1}{2}\delta t^2 \left.\frac{\partial^2 u}{\partial t^2}\right|_j^n + \ldots \approx \frac{c\delta t}{2\delta x}\left(2\delta x \left.\frac{\partial u}{\partial x}\right|_j^n + \tfrac{1}{3}\delta x^3 \left.\frac{\partial^3 u}{\partial x^3}\right|_j^n + \ldots\right. \tag{2.11}$$

so that, when the original differential equation (2.6) is subtracted, we are left with a truncation error which is 2nd order in the time but 3rd order in the spatial part.

The *stability* is a property of the time, t, integration rather than the space, x. We analyse this by considering a plane wave solution for the x–dependence by substituting $u_j^n = v^n \exp(\mathrm{i}kx_j)$ to obtain

$$v^{n+1}e^{\mathrm{i}kx_j} = v^n e^{\mathrm{i}kx_j} - \frac{c\delta t}{2\delta x}v^n\left(e^{ikx_{j+1}} - e^{\mathrm{i}kx_{j-1}}\right) \tag{2.12}$$

or, after dividing out the common exponential factor,

$$v^{n+1} = \left[1 - \mathrm{i}\frac{c\delta t}{\delta x}\sin(k\delta x)\right]v^n. \tag{2.13}$$

Since the wave (2.3) and advection (2.6) equations express a conservation law (section 2.6) the solution should neither grow nor decay as a function of time. If we substitute $v^n \mapsto v^n + \delta v^n$ and subtract (2.13) we obtain an equation for δv^n

$$\delta v^{n+1} = \left[1 - \mathrm{i}\frac{c\delta t}{\delta x}\sin(k\delta x)\right]\delta v^n \tag{2.14}$$

which is identical to (2.13). Hence the stability condition is simply given by the quantity in square brackets. We call this the *amplification factor* and write it as

$$g = 1 - \mathrm{i}\alpha \tag{2.15}$$

where

$$\alpha = \frac{c\delta t}{\delta x}\sin(k\delta x). \tag{2.16}$$

As g is complex the stability condition becomes

$$|g|^2 = 1 + \alpha^2 \leq 1 \text{ for all } k\,. \tag{2.17}$$

The condition must be fulfilled for all wave vectors k; otherwise a component with a particular k will tend to grow at the expense of the others. This is known as the *von Neumann* stability condition[2]. Unfortunately it is never fulfilled for the simple method applied to the advection equation: i.e. the method is unstable for the advection equation.

2.3.2 An Improved Algorithm — the Lax method

We see from (2.17) that our simple method is in fact unstable for the advection equation, for all finite values of δx and δt. How might we improve on this? Let us consider a minor (?) modification of (2.8)

$$u_j^{n+1} = \tfrac{1}{2}\left(u_{j+1}^n + u_{j-1}^n\right) - \frac{c\delta t}{2\delta x}\left(u_{j+1}^n - u_{j-1}^n\right). \tag{2.18}$$

in which the term in u_j^n has been replaced by an average over its 2 neighbours. When we apply the same (*von Neumann*) analysis to this algorithm we find

$$v^{n+1} = \left[\cos(k\delta x) - i\frac{c\delta t}{\delta x}\sin(k\delta x)\right]v^n \tag{2.19}$$

so that

$$|g|^2 = \cos^2(k\delta x) + \left(\frac{c\delta t}{\delta x}\right)^2\sin^2(k\delta x) \tag{2.20a}$$

$$= 1 - \sin^2(k\delta x)\left\{1 - \left(\frac{c\delta t}{\delta x}\right)^2\right\} \tag{2.20b}$$

[2] von Neumann is perhaps better known for his work on the theory of computing, but his name is also associated with various other aspects of mathematical physics

which is stable for all k as long as

$$\frac{\delta x}{\delta t} \geq c \tag{2.21}$$

which is an example of the *Courant–Friedrichs–Lewy* condition applicable to hyperbolic equations.

There is a simple physical explanation of this condition: if we start with the initial condition such that $u(x, t = 0) = 0$ everywhere except at one point on the spatial grid, then a point m steps away on the grid will remain zero until at least m time steps later. If, however, the equation is supposed to describe a physical phenomenon which travels faster than that then something must go wrong. This is equivalent to the condition that the time step, δt, must be smaller than the time taken for the wave to travel the distance of the spatial step, δx; or that the speed of propagation of information on the grid, $\delta x/\delta t$, must be greater than any other speed in the problem.

2.3.3 Non–Linear Equations

When applying the von Neumann analysis to non–linear equations the substitution $y_i \mapsto y_i + \delta y_i$ should be performed before setting $\delta y_i = \delta y \exp(\mathrm{i}kx_i)$. y_i should then be treated as a constant, independent of x (or i). The resulting stability condition will then depend on y, just as in the case of ODE's (section 1.2.4).

2.3.4 Other methods for Hyperbolic Equations

There is a large number of algorithms applicable to hyperbolic equations in general. As before, their suitability for solving a particular problem must be based on an analysis of their accuracy and stability for any wavelength. Here we simply name a few of them:

- The Lelevier method.
- The one–step and two–step Lax–Wendroff methods.
- The Leap–Frog method.
- The Quasi–Second Order method.

These are discussed in more detail in chapter II of Potter (1973) and chapter 17 of Press et al. (1989).

2.4 Eulerian and Lagrangian Methods

In the methods discussed so far the differential equations have been discretised by defining the value of the unknown at fixed points in space. Such methods are known as *Eulerian* methods. We have confined ourselves to a regular grid of points, but sometimes it is advantageous to choose some other grid. An obvious example is when there is some symmetry in the problem, such as cylindrical or spherical: often it is appropriate to base our choice of grid on cylindrical or spherical coordinates rather than the Cartesian ones used so far.

Suppose, however, we are dealing with a problem in electromagnetism or optics, in which the dielectric constant varies in space. Then it might be appropriate to choose a grid in which the points are more closely spaced in the regions of higher dielectric constant. In that way we could take account of the fact that the wavelengths expected in the solution will be shorter. Such an approach is known as an *adaptive grid.*

In fact it is not necessary for the grid to be fixed in space. In fluid mechanics, for example, it is often better to define a volume of space containing a fixed mass of fluid and to let the boundaries of these cells move in response to the dynamics of the fluid. The differential equation is transformed into a form in which the variables are the positions of the boundaries of the cells rather than the quantity of fluid in each cell. A simple example of such a *Lagrangian* method is described in the Lagrangian Fluid project (section 2.9).

2.5 Parabolic Equations — Diffusion

Parabolic equations such as the *diffusion* and *time–dependent Schrödinger* equations are similar to the hyperbolic case in that the boundary is open and we consider integration with respect to time, but, as we shall see, they present some extra problems.

2.5.1 A Simple Method

We consider the diffusion equation and apply the same simple method we tried for the hyperbolic case.

$$\frac{\partial u}{\partial t} - \kappa \frac{\partial^2 u}{\partial x^2} = 0 \tag{2.22}$$

and discretise it using the Euler method for the time derivative and the simplest centred 2nd order derivative to obtain

$$u_j^{n+1} = u_j^n + \frac{\kappa \delta t}{\delta x^2} \left(u_{j+1}^n - 2u_j^n + u_{j-1}^n \right). \tag{2.23}$$

Applying the *von Neumann* analysis to this system by considering a single Fourier mode in x space, we obtain

$$v^{n+1} = v^n \left[1 - \frac{4\kappa \delta t}{\delta x^2} \sin^2 \left(\frac{k \delta x}{2} \right) \right] \tag{2.24}$$

so that the condition that the method is stable for all k gives

$$\delta t \leq \tfrac{1}{2} \frac{\delta x^2}{\kappa}. \tag{2.25}$$

Although the method is in fact conditionally stable the condition (2.25) hides an uncomfortable property: namely, that if we want to improve accuracy and allow for smaller wavelengths by halving δx we must divide δt by 4. Hence, the number of space steps is doubled and the number of time steps is quadrupled: the time required is multiplied by 8. Note that this is different from the sorts of conditions we have encountered up to now, in that it doesn't depend on any real physical time scale of the problem.

2.5.2 The Dufort–Frankel Method

We consider here one of many alternative algorithms which have been designed to overcome the stability problems of the simple algorithm. The *Dufort–Frankel* method is a trick which exploits the unconditional stability of the intrinsic method (section 1.6) for simple differential equations.

We modify (2.23) to read

$$u_j^{n+1} = u_j^{n-1} + \frac{2\kappa \delta t}{\delta x^2} \left\{ u_{j+1}^n - \left(u_j^{n+1} + u_j^{n-1} \right) + u_{j-1}^n \right\}. \tag{2.26}$$

which can be solved explicitly for u_j^{n+1} at each mesh point

$$u_j^{n+1} = \left(\frac{1-\alpha}{1+\alpha} \right) u_j^{n-1} + \left(\frac{\alpha}{1+\alpha} \right) \left(u_{j+1}^n + u_{j-1}^n \right) \tag{2.27}$$

where

$$\alpha = 2 \frac{\kappa \delta t}{\delta x^2}. \tag{2.28}$$

When the usual *von Neumann* analysis is applied to this method it is found to be unconditionally stable. Note however that this does not imply that δx and δt can be made indefinitely large; common sense tells us that they must be small compared to any real physical time or length scales in the problem. We must still worry about the accuracy of the method. Another difficulty this method shares with the Leap–Frog (section 1.3) method is that it requires boundary conditions at 2 times rather than one, even though the original diffusion equation is only 1st order in time.

2.5.3 Other Methods

Another important method for solving parabolic equations is the *Crank–Nicholson* method, which we shall not discuss here but which is considered in chapter II of Potter (1973).

2.6 Conservative Methods

Most of the differential equations we encounter in physics embody some sort of conservation law. It is important therefore to try to ensure that the method chosen to solve the equation obeys the underlying conservation law exactly and not just approximately. This principle can also have the side effect of ensuring stability. The simplest approach to deriving a method with such properties is to go back to the original derivation of the differential equation.

2.6.1 The Equation of Continuity

The archetypal example of a differential equation which implies a conservation law is the equation of continuity, which, in its differential form says that

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \boldsymbol{j} = 0 \tag{2.29}$$

where ρ represents a density and $\boldsymbol{j}$ a current density. The density and current density could be of mass, charge, energy or something else which is conserved. Here we shall use the words *charge* and *current* for convenience. We consider here the 1D form for simplicity. The equation is derived by considering space as divided into sections of length δx. The change in the total *charge* in a section is equal to the total current coming in (going out) through its ends

$$\frac{\partial}{\partial t}\int_{x_i}^{x_{i+1}} \rho \, \mathrm{d}x = -\int \boldsymbol{j} \cdot \mathrm{d}S_i, \tag{2.30}$$

It is therefore useful to re–express the differential equation in terms of the total charge in the section and the total current coming in through each face, so that we obtain a discrete equation of the form

$$\frac{\partial}{\partial t} Q_j = \boldsymbol{I}_{j-1,j} - \boldsymbol{I}_{j,j+1} \tag{2.31}$$

where Q_j represents the total *charge* in part j and $\boldsymbol{I}_{j,j+1}$ is the current through the boundary between parts j and $j+1$.

This takes care of the spatial part. What about the time derivative? We can express the physics thus:

> *The change in the charge in a cube is equal to the total charge which enters through its faces.*

2.6.2 The Diffusion Equation

In many cases the current through a face is proportional to the difference in density (or total charge) between neighbouring cubes

$$\boldsymbol{I}_{j,j+1} = -\beta \left(Q_{j+1} - Q_j\right). \tag{2.32}$$

Substituting this into the equation of continuity leads directly to the diffusion equation in discrete form

$$Q_j^{n+1} = Q_j^n + \delta t \beta \left(Q_{j+1}^n - 2Q_j^n + Q_{j-1}^n\right) \tag{2.33}$$

which is of course our simple method (2.23) of solution.

To check whether this algorithm obeys the conservation law we sum over all j, as $\sum_j Q_j$ should be conserved. Note that it helps to consider the whole process as taking place on a circle as this avoids problems associated with currents across the boundaries. In this case (e.g.) $\sum_j Q_{j+1} = \sum_j Q_j$ and it is easy to see that the conservation law is obeyed for (2.33).

2.6.3 Maxwell's Equations

Let us now try to apply the same ideas to Maxwell's equations. In free space we have

$$\nabla \times \boldsymbol{E} = -\mu_0 \frac{\partial \boldsymbol{H}}{\partial t} \tag{2.34a}$$

$$\nabla \times \boldsymbol{H} = +\epsilon_0 \frac{\partial \boldsymbol{E}}{\partial t}. \tag{2.34b}$$

In order to reverse the derivation of these equations we consider space to be divided into cubes as before. For (2.34a) we integrate over a face of the cube and apply Stokes' theorem

$$\oint_S \nabla \times \boldsymbol{E} \cdot \mathrm{d}\boldsymbol{S} = \int \boldsymbol{E} \cdot \mathrm{d}\boldsymbol{l} \tag{2.35a}$$

$$= -\frac{\partial}{\partial t} \oint_S \mu_0 \boldsymbol{H} \cdot \mathrm{d}\boldsymbol{S} \tag{2.35b}$$

and the integral of the electric field, $\boldsymbol{E}$, round the edges of the face is equal to minus the rate of change of the magnetic flux through the face, i.e. Faraday's law. Here we can associate the electric field, $\boldsymbol{E}$ with the edges and the magnetic field with the face of the cube.

In the case of the diffusion equation we had to think in terms of the total *charge* in a cube instead of the density. Now we replace the electric field with the integral of the field along a line and the magnetic field with the flux through a face.

Note also that we can analyse (2.34b) in a similar way to obtain a representation of Ampère's law.

2.7 Dispersion

Let us return to the Lax method (section 2.3.2) for hyperbolic equations. The solution of the differential equation has the form

$$u(x,t) = u_0 e^{\mathrm{i}(\omega t - kx)} \tag{2.36}$$

where $\omega = ck$. Let us substitute (2.36) into the Lax algorithm

$$v e^{\mathrm{i}(\omega t^{n+1} - kx_j)} = \tfrac{1}{2} v e^{\mathrm{i}(\omega t^n - kx_j)} \left\{ \left(e^{+\mathrm{i}k\delta x} + e^{-\mathrm{i}k\delta x}\right) - \frac{c\delta t}{\delta x} \left(e^{+\mathrm{i}k\delta x} - e^{-\mathrm{i}k\delta x}\right) \right\}. \tag{2.37}$$

By cancelling the common factors we now obtain

$$e^{\mathrm{i}\omega\delta t} = \cos(k\delta x) - \mathrm{i}\frac{c\delta t}{\delta x} \sin(k\delta x). \tag{2.38}$$

From this we can derive a dispersion relation, ω vs k, for the discrete equation and compare the result with $\omega = ck$ for the original differential equation. Since, in general, ω could be complex we write it as $\omega = \Omega + \mathrm{i}\gamma$ and compare real and imaginary parts on both sides of the equation to obtain

$$\cos \Omega\delta t e^{-\gamma\delta t} = \cos k\delta x \tag{2.39a}$$

$$\sin \Omega\delta t e^{-\gamma\delta t} = -\frac{c\delta t}{\delta x} \sin k\delta x. \tag{2.39b}$$

Taking the ratio of these or the sum of their squares respectively leads to the equations

$$\tan(\Omega\delta t) = \frac{c\delta t}{\delta x} \tan(k\delta x) \tag{2.40a}$$

$$e^{-2\gamma\delta t} = \cos^2(k\delta x) + \left(\frac{c\delta t}{\delta x}\right)^2 \sin^2(k\delta x). \tag{2.40b}$$

The first of these equations tells us that in general the phase velocity is not c, although for long wavelengths, $k\delta x \ll 1$ and $\Omega\delta t \ll 1$, we recover the correct dispersion relationship. This is similar to the situation when

we compare lattice vibrations with classical elastic waves: the long wavelength *sound* waves are OK but the shorter wavelengths deviate.

The second equation (2.40b) describes the damping of the modes. Again for small $k\delta x$ and $\gamma\delta t$, $\gamma = 0$, but (e.g.) short wavelengths, $\lambda = 4\delta x$, are strongly damped. This may be a desirable property as short wavelength oscillations may be spurious. After all we should have chosen δx to be small compared with any expected features. Nevertheless with this particular algorithm $\lambda = 2\delta x$ is not damped. Other algorithms, such as Lax–Wendroff, have been designed specifically to damp anything with a $k\delta x > \frac{1}{4}$.

2.8 Problems

1. Explain the difference between hyperbolic, parabolic and elliptic partial differential equations, and give an example of each. What is the important physical distinction between hyperbolic and parabolic equations, on the one hand, and elliptic equations on the other?

2. Describe the von Neumann procedure for analysing the stability of partial differential equations.

3. Describe the physical principle behind the Courant–Friedrichs–Lewy condition as applied to the numerical solution of partial differential equations.

4. Are the following equations hyperbolic, elliptic or parabolic?

   a) $$3\frac{\partial^2 f}{\partial t^2} + 2\frac{\partial^2 f}{\partial x^2} - \frac{\partial^2 f}{\partial t \partial x} = 0$$

   b) $$7\frac{\partial f}{\partial x} + 3\frac{\partial^2 f}{\partial t^2} - \frac{\partial^2 f}{\partial t \partial x} - 2\frac{\partial^2 f}{\partial x^2} = 0$$

   c) $$3\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial t^2} + 6\frac{\partial^2 f}{\partial t \partial x} + \frac{\partial f}{\partial t} = e^x$$

   d) $$\frac{\partial^2 f}{\partial t^2} + \frac{\partial^2 f}{\partial x^2} - 2\frac{\partial^2 f}{\partial t \partial x} + \frac{\partial f}{\partial x} + 2\frac{\partial f}{\partial t} = 0$$

   e) $$\frac{\partial}{\partial t}\left(\frac{\partial f}{\partial t} + 3\frac{\partial f}{\partial x} - f\right) + \frac{\partial}{\partial x}\left(2\frac{\partial f}{\partial t} - \frac{\partial f}{\partial x}\right) = \sin(x + t^3)$$

5. The equation
   $$\frac{\partial f}{\partial t} = \beta\frac{\partial^3 f}{\partial x^3}$$
   can be represented by the difference equation
   $$f_n^{(m+1)} = f_n^{(m)} + \mu\left(f_{n+2}^{(m)} - 3f_{n+1}^{(m)} + 3f_n^{(m)} - f_{n-1}^{(m)}\right) \qquad \mu = \frac{\beta\delta t}{\delta x^3}.$$
   Derive the truncation error of this difference equation.

   Write down an alternative difference equation which is 2nd order accurate in both δt and δx.

6. The Dufort–Frankel (2.26) scheme is a method for the solution of the diffusion equation.

   Show that the method is unconditionally stable.

   Discuss the advantages and disadvantages of this method.

7. The diffusion equation in a medium where the diffusion constant D varies in space ($D = D(x)$) is
   $$\frac{\partial f}{\partial t} = \frac{\partial}{\partial x}\left(D\frac{\partial f}{\partial x}\right) \qquad \text{or} \qquad \frac{\partial f}{\partial t} = D\frac{\partial^2 f}{\partial x^2} + \frac{\partial D}{\partial x}\frac{\partial f}{\partial x}.$$
   Show that the difference equation
   $$\frac{f_n^{(m+1)} - f_n^{(m)}}{\delta t} = D_n\frac{f_{n+1}^{(m)} - 2f_n^{(m)} + f_{n-1}^{(m)}}{\delta x^2} + \left(\frac{D_{n+1} - D_{n-1}}{2\delta x}\right)\left(\frac{f_{n+1}^{(m)} - f_{n-1}^{(m)}}{2\delta x}\right)$$

is not conservative, i.e. $\int f \mathrm{d}x$ is not conserved.

Construct an alternative difference scheme which is conservative.

8. Show that the Lax (2.18) scheme for the solution of the advection equation is equivalent to

$$\frac{\partial f}{\partial t} = -u\frac{\partial f}{\partial x} + \left(\frac{\delta x^2}{2\delta t} - \tfrac{1}{2}u^2\delta t\right)\frac{\partial^2 f}{\partial x^2} + \text{higher order terms.}$$

Examine the behaviour of wave–like solutions $f = \exp(\mathrm{i}(kx - \omega t))$ in the Lax scheme and explain the behaviour in terms of diffusion.

9. Describe what is meant by *numerical dispersion.*

10. Lax–Wendroff method consists of 2 steps, just like Runge–Kutta or Predictor–Corrector. It is given by

$$u_{j+1/2}^{n+1/2} = \tfrac{1}{2}\left(u_{j+1}^{n} + u_j^n\right) - \tfrac{1}{2}c\frac{\delta t}{\delta x}\left(u_{j+1}^n - u_j^n\right) \tag{2.41}$$

$$u_j^{n+1} = u_j^n - c\frac{\delta t}{\delta x}\left(u_{j+1/2}^{n+1/2} - u_{j-1/2}^{n+1/2}\right) \tag{2.42}$$

Draw a diagram to illustrate the way this algorithm operates on an (x, t) grid.

Show that the algorithm is stable provided the Courant–Friedrichs–Lewy condition is obeyed.

Show that the algorithm tends to dampen waves with wavelength $\lambda = 2\delta x$.

2.9 Project — Lagrangian Fluid Code

Small amplitude sound waves travel at speed $c = \sqrt{\gamma P/\rho}$ without changing shape (so that a sine wave remains a sine wave). However, when the amplitude is large, the sound speed differs between the peaks and the troughs of the wave and the wave changes shape as it propagates. This project is to simulate this effect in one-dimension. A difference method is used in which the system is split up into cells and the cell boundaries are allowed to move with the local fluid velocity (a so–called Lagrangian (section—2.4) method). You should not require any library routines to complete this project.

2.9.1 The Difference Equations

The difference equations are,

$$u_{n+\frac{1}{2}}^{(m+\frac{1}{2})} = u_{n+\frac{1}{2}}^{(m-\frac{1}{2})} + (P_n^{(m)} - P_{n+1}^{(m)})\delta t/\Delta m \tag{2.43a}$$

$$x_{n+\frac{1}{2}}^{(m+1)} = x_{n+\frac{1}{2}}^{(m)} + u_{n+\frac{1}{2}}^{(m+\frac{1}{2})}\delta t \quad , \tag{2.43b}$$

where,

$$P_n^{(m)} = \text{constant} \times (\rho_n^{(m)})^\gamma \quad . \tag{2.44}$$

and,

$$\rho_n^{(m)} = \Delta m/(x_{n+\frac{1}{2}}^{(m)} - x_{n-\frac{1}{2}}^{(m)}) \quad , \tag{2.45}$$

$x_{n+\frac{1}{2}}^{(m)}$ and $u_{n+\frac{1}{2}}^{(m)}$ are the positions and velocities of the cell boundaries. $\rho_n^{(m)}$ and $P_n^{(m)}$ are the densities and pressures in the cells. Δm is the mass in each cell.

It is useful for the purpose of programming to redefine things to get rid of the various half integer indices. Hence we can write the equations as

$$u_n'^{(m+1)} = u_n'^{(m)} + (P_n^{(m)} - P_{n+1}^{(m)})\delta t/\Delta m \tag{2.46a}$$

$$x_n'^{(m+1)} = x_n'^{(m)} + u_n'^{(m+1)}\delta t \tag{2.46b}$$

$$\rho_n^{(m)} = \Delta m/(x_n'^{(m)} - x_{n-1}'^{(m)}) \quad , \tag{2.46c}$$

which maps more easily onto the arrays in a program.

2.9.2 Boundary Conditions

Use periodic boundary conditions to allow simulation of an infinite wave train. Note, though, the presence of P_{n+1} and x_{n-1} in (2.46). If n runs from 1 to N then P_{N+1} and x_0 will be needed. This is best done by creating "ghost" cells at the end of the arrays. In C(++) this can be done by a declaration such as

```
double x[N+1], u[N+1], P[N+2];
```

and, after each loop in which P or x are updated, by setting $P_{N+1} = P_1$ and $x_0 = x_N - L$, where L is the equilibrium length of the grid. Note that this requires you to leave a couple of empty array elements, but for large N this is not significant.

2.9.3 Initial Conditions

Sensible initial conditions might be,

$$u_n^{\prime(0)} = Mc\sin(2\pi n/N + \pi c\delta t/L) \tag{2.47a}$$
$$x_n^{\prime(0)} = nL/N + (ML/2\pi)\cos(2\pi n/N) \quad , \tag{2.47b}$$

where M, the amplitude of the sound wave, is the ratio of u to the sound speed (i.e. the Mach number). Be careful to choose sensible values for the parameters: e.g. the values of x_n should rise monotonically with n, otherwise some cells will have negative fluid densities.

The essential physics of the problem is independent of the absolute values of the equilibrium pressure and density so you can set $L = 1$, $\Delta m = (1/N)$ and $P = \rho^\gamma$. In addition you can assume that $\gamma = C_\mathrm{p}/C_\mathrm{v} = \frac{5}{3}$.

Note that the scheme is *leap-frog* but not of the dangerous variety: alternate steps solve for u and x.

2.9.4 The Physics

Start with a relatively small value for M ($\ll 1$) and show that the wave maintains its shape and moves at the correct velocity. Then increase M to find out what happens. The stability analysis for a non–linear equation like this is difficult. Try halving the distance between grid points. How does the behaviour of the system change? Do the effects you observe describe real physics or are they numerical artefacts?

One common numerical problem only manifests itself for large M. Try running for a few steps at $M = 1$, look at the values for x and try to work out what has gone wrong. Think about how to prevent this problem (Hint: you should find something very similar to the Courant–Friedrichs-Lewy condition).

2.9.5 An Improvement?

Consider the operation:

$$y_n \mapsto \tfrac{1}{4}\left(y_{n+1} + 2y_n + y_{n-1}\right) \tag{2.48}$$

By applying the von Neumann trick you should be able to deduce the effect of such an operation. Now try applying the transformation (2.48) to to x and u after a few iterations. How does the behaviour of your simulation change?

2.9.6 The Report

In your report you should discuss the behaviour of the simulation, being careful to distinguish between real physics and numerical artefacts. Discuss what measures you employed to control the numerical problems and explain the reasoning behind them.

2.10 Project — Solitons

2.10.1 Introduction

The Korteweg de Vries equation,

$$\frac{\partial^3 y}{\partial x^3} + y\frac{\partial y}{\partial x} + \frac{\partial y}{\partial t} = 0 \quad , \tag{2.49}$$

is one of a class of non–linear equations which have so–called soliton solutions. In this case a solution can be written in the form,

$$y = 12\alpha^2 \operatorname{sech}^2\left[\alpha(x - 4\alpha^2 t)\right] \quad , \tag{2.50}$$

which has the form of a pulse which moves unchanged through the system. Ever since the phenomenon was first noted (on a canal in Scotland) it has been recognised in a wide range of different physical situations. The "bore" which occurs on certain rivers, notably the Severn, is one such. The dynamics of phase boundaries in various systems, such as domain boundaries in a ferromagnet, and some meteorological phenomena can also be described in terms of solitons.

2.10.2 Discretisation

The simplest discretisation of (2.49), based on the Euler method, gives the equation

$$y_j^{(n+1)} = y_j^{(n)} - \left[\tfrac{1}{4}\frac{\delta t}{\delta x}\left\{\left(y_{j+1}^{(n)}\right)^2 - \left(y_{j-1}^{(n)}\right)^2\right\} + \tfrac{1}{2}\frac{\delta t}{\delta x^3}\left\{y_{j+2}^{(n)} - 2y_{j+1}^{(n)} + 2y_{j-1}^{(n)} - y_{j-2}^{(n)}\right\}\right]. \tag{2.51}$$

You should check that this is indeed a sensible discretisation of (2.49) but that it is unstable. Note that when analysing the non–linear term in (2.51) you should make the substitution $y \mapsto y + \delta y$ and retain the linear terms in δy. Thereafter you should treat y as a constant, independent of x and t, and apply the von Neumann method to δy.

If you find the full stability analysis difficult you might consider the 2 limits of large and small y. In the former case the 3rd derivative is negligible and (2.49) reduces to a non–linear advection equation, whereas in the latter the non–linear term is negligible and the equation is similar to the diffusion equation but with a 3rd derivative. In any case you will require to choose δt so that the equation is stable in both limits.

You are free to choose any method you wish to solve the equation but you will find the Runge–Kutta or Predictor–Corrector methods most reliable. Hence treating y_j as a long vector $\mathbf{y}$ and the terms on the right–hand–side of (2.51) as a vector function $\mathbf{f}(\mathbf{y})$ the R–K method can be written concisely as

$$\mathbf{y}'^{(n+1/2)} = \mathbf{y}^{(n)} - \tfrac{1}{2}\delta t\,\mathbf{f}(\mathbf{y}^{(n)}) \tag{2.52a}$$

$$\mathbf{y}^{(n+1)} = \mathbf{y}^{(n)} - \delta t\,\mathbf{f}(\mathbf{y}'^{(n+1/2)}) \quad , \tag{2.52b}$$

where $\delta t\,\mathbf{f}(\mathbf{y})$ is the quantity in square brackets [] in (2.51). Check that this method is stable, at least in the 2 limiting cases. Bear in mind that the Runge–Kutta method is usable for oscillatory equations in spite of the small instability as long as the term $(\omega\delta t)^4$ is small.

By studying the analytical solution (2.50) you should be able to choose a sensible value for δx in terms of α and from the stability conditions you can deduce an appropriate δt. Again, by looking at (2.50) you should be able to decide on a sensible size for the total system.

You should use periodic boundary conditions, so that your solitons can run around your system several times if necessary. The easiest way to do this is by using "ghost" elements at each end of your arrays. Suppose your arrays should run from y_1 to y_N. Then you can add a couple of extra elements to each end: $y_{-1}, y_0, y_{N+1}, y_{N+2}$. After each step you can then assign these values as

$$y_{-1} = y_{N-1}, \quad y_0 = y_N, \quad y_{N+1} = y_1, \quad y_{N+2} = y_2 \tag{2.53}$$

so that the derivative terms in (2.51) can be calculated without having to take any special measures.

2.10.3 Physics

You should first check your program by studying the single soliton case. Use (2.50) at $t = 0$ as your initial condition and watch how the soliton develops. Check that after it has gone round the system once it retains its shape.

Now you can try 2 solitons of different sizes. The initial conditions should be the simple sum of 2 solitons well separated from one another. Watch what happens when the solitons meet.

In your report you should discuss the behaviour of the solitons as well as the properties of the method chosen to solve the equations.

Computational Physics
Matrix Algebra

3.1 Introduction

Nearly every scientific problem which is solvable on a computer can be represented by matrices. However the ease of solution can depend crucially on the types of matrices involved. There are 3 main classes of problems which we might want to solve:

1. Trivial Algebraic Manipulation such as addition, $\mathbf{A} + \mathbf{B}$ or multiplication, $\mathbf{AB}$, of matrices.
2. Systems of equations: $\mathbf{A}\boldsymbol{x} = \boldsymbol{b}$ where $\mathbf{A}$ and $\boldsymbol{b}$ are known and we have to find $\boldsymbol{x}$. This also includes the case of finding the inverse, $\mathbf{A}^{-1}$. The standard example of such a problem is Poisson's equation (section 3.4).
3. Eigenvalue Problems: $\mathbf{A}\boldsymbol{x} = \alpha\boldsymbol{x}$. This also includes the generalised eigenvalue problem: $\mathbf{A}\boldsymbol{x} = \alpha\mathbf{B}\boldsymbol{x}$. Here we will consider the time–independent Schrödinger equation.

In most cases there is no point in writing your own routine to solve such problems. There are many computer libraries, such as *Numerical Algorithms Group* (n.d.), *Lapack Numerical Library* (n.d.) (for linear algebra problems and eigenvalue problems). which contain well tried routines. In addition vendors of machines with specialised architectures often provide libraries of such routines as part of the basic software.

3.2 Types of Matrices

There are several ways of classifying matrices depending on symmetry, sparsity etc. Here we provide a list of types of matrices and the situation in which they may arise in physics.

- **Hermitian Matrices:** Many Hamiltonians have this property especially those containing magnetic fields: $A_{ji} = A_{ij}^{\dagger}$ where at least some elements are complex.
- **Real Symmetric Matrices:** These are the commonest matrices in physics as most Hamiltonians can be represented this way: $A_{ji} = A_{ij}$ and all A_{ij} are real. This is a special case of Hermitian matrices.
- **Positive Definite Matrices:** A special sort of Hermitian matrix in which all the eigenvalues are positive. The overlap matrices used in *tight–binding* electronic structure calculations are like this. Sometimes matrices are *non–negative definite* and zero eigenvalues are also allowed. An example is the dynamical matrix describing vibrations of the atoms of a molecule or crystal, where $\omega^2 \geq 0$.
- **Unitary Matrices:** The product of the matrix and its Hermitian conjugate is a unit matrix, $\mathbf{U}^{\dagger}\mathbf{U} = \mathbf{I}$. A matrix whose columns are the eigenvectors of a Hermitian matrix is unitary; the unitarity is a consequence of the orthogonality of the eigenvectors. A scattering ($\mathbf{S}$) matrix is unitary; in this case a consequence of current conservation.

- **Diagonal Matrices:** All matrix elements are zero except the diagonal elements, $A_{ij} = 0$ when $i \neq j$. The matrix of eigenvalues of a matrix has this form. Finding the eigenvalues is equivalent to *diagonalisation*.
- **Tridiagonal Matrices:** All matrix elements are zero except the diagonal and first off diagonal elements, $A_{i,i} \neq 0$, $A_{i,i\pm1} \neq 0$. Such matrices often occur in 1 dimensional problems and at an intermediate stage in the process of diagonalisation.
- **Upper and Lower Triangular Matrices:** In Upper Triangular Matrices all the matrix elements below the diagonal are zero, $A_{ij} = 0$ for $i > j$. A Lower Triangular Matrix is the other way round, $A_{ij} = 0$ for $i < j$. These occur at an intermediate stage in solving systems of equations and inverting matrices.
- **Sparse Matrices:** Matrices in which most of the elements are zero according to some pattern. In general sparsity is only useful if the number of non–zero matrix elements of an $N \times N$ matrix is proportional to N rather than N^2. In this case it may be possible to write a function which will multiply a given vector $\boldsymbol{x}$ by the matrix $\mathbf{A}$ to give $\mathbf{A}\boldsymbol{x}$ without ever having to store all the elements of $\mathbf{A}$. Such matrices commonly occur as a result of simple discretisation of partial differential equations (see chapter 1.9.4), and in simple models to describe many physical phenomena.
- **General Matrices:** Any matrix which doesn't fit into any of the above categories, especially non–square matrices.

There are a few extra types which arise less often:

- **Complex Symmetric Matrices:** Not generally a useful symmetry. There are however two related situations in which these occur in theoretical physics: Green's functions and scattering ($\mathbf{S}$) matrices. In both these cases the real and imaginary parts commute with each other, but this is not true for a general complex symmetric matrix.
- **Symplectic Matrices:** This designation is used in 2 distinct situations:
  - The eigenvalues occur in pairs which are reciprocals of one another. A common example is a Transfer Matrix.
  - Matrices whose elements are *Quaternions*, which are 2×2 matrices like

$$\begin{pmatrix} a & b \\ -b^* & a^* \end{pmatrix}. \tag{3.1}$$

    Such matrices describe systems involving spin–orbit coupling. All eigenvalues are doubly degenerate (*Kramers degeneracy*).

3.3 Simple Matrix Problems

3.3.1 Addition and Subtraction

In programming routines to add or subtract matrices it pays to remember how the matrix is stored in the computer. Unfortunately this varies from language to language: in C(++) and Pascal the first index varies slowest and the matrix is stored one complete *row* after another; whereas in FORTRAN the first index varies fastest and the matrix is stored one complete *column* after another. It is generally most efficient to access the matrix elements in the order in which they are stored. Hence the simple matrix algebra, $\mathbf{A} = \mathbf{B} + \mathbf{C}$, should be written in C[1] as

[1] In C(++) it is usual to use `double` instead of `float` when doing numerical programming. In most cases single precision is not accurate enough.

```
const int N =??;
int i, j;
double A[N][N], B[N][N], C[N][N];
for( i = 0; i < N; i++)
   for( j = 0; j < N; j++)
      A[i][j] = B[i][j] + C[i][j];
```

or its equivalent using pointers. In FORTRAN90 this should be written

```
integer :: i,j
integer, parameter :: N = ??
real, double precision :: A(N,N), B(N,N), C(N,N)
do i = 1,N
   do j=1,N
      A(j,i) = B(j,i) + C(j,i)
   end do
end do
```

Note the different ordering of the loops in these 2 examples. This is intended to optimise the order in which the matrix elements are accessed. It is perhaps worth noting at this stage that in both C++ and FORTRAN90 it is possible to define matrix type variables (classes) so that the programs could be reduced to

```
matrix A(N,N), B(N,N), C(N,N);
A = B + C;
```

The time taken to add or subtract 2 $M \times N$ matrices is generally proportional to the total number of matrix elements, MN, although this may be modified by parallel architecture.

3.3.2 Multiplication of Matrices

Unlike addition and subtraction of matrices it is difficult to give a general *machine independent* rule for the optimum algorithm for the operation. In fact matrix multiplication is a classic example of an operation which is very dependent on the details of the architecture of the machine. We quote here a general purpose example but it should be noted that this does not necessarily represent the optimum ordering of the loops.

```
const int L = ??, M = ??, N = ??;
int i, j, k;
double A[L][N], B[L][M], C[M][N], sum;
for ( j = 0; j < N; i++)
   for ( i = 0; i < L; j++)
   {
      for ( sum = 0, k = 0; k < M; k++)
         sum += B[i][k] * C[k][j];
      A[i][j] = sum;
   }
```

in C.

The time taken to multiply 2 $N \times N$ matrices is generally proportional to N^3, although this may be modified by parallel processing.

3.4 Elliptic Equations — Poisson's Equation

At this point we digress to discuss Elliptic Equations such as Poisson's equation. In general these equations are subject to boundary conditions at the outer boundary of the range; there are no *initial* conditions, such as we would expect for the Wave or Diffusion equations. Hence they cannot be solved by adapting the methods for simple differential equations.

3.4.1 One Dimension

We start by considering the one dimensional Poisson's equation

$$\frac{\mathrm{d}^2V}{\mathrm{d}x^2} = f(x). \tag{3.2}$$

The 2nd derivative may be discretised in the usual way to give

$$V_{n-1} - 2V_n + V_{n+1} = \delta x^2 \cdot f_n \tag{3.3}$$

where we define $f_n = f(x = x_n = n\delta x)$.

The boundary conditions are usually of the form $V(x) = V_0$ at $x = x_0$ and $V(x) = V_{N+1}$ at $x = x_{N+1}$, although sometimes the condition is on the first derivative. Since V_0 and V_{N+1} are both known the $n = 1$ and $n = N$ equations (3.3) may be written as

$$-2V_1 + V_2 = \delta x^2 \cdot f_1 - V_0 \tag{3.4a}$$

$$V_{N-1} - 2V_N = \delta x^2 \cdot f_N - V_{N+1}. \tag{3.4b}$$

This may seem trivial but it maintains the convention that all the terms on the left contain unknowns and everything on the right is known. It also allows us to rewrite the (3.3) in matrix form as

$$\begin{bmatrix} -2 & 1 & & & & & & \\ 1 & -2 & 1 & & & & & \\ & 1 & -2 & 1 & & & & \\ & & \ddots & \ddots & \ddots & & & \\ & & & 1 & -2 & 1 & & \\ & & & & \ddots & \ddots & \ddots & \\ & & & & & 1 & -2 & 1 \\ & & & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ \vdots \\ V_n \\ \vdots \\ V_{N-1} \\ V_N \end{bmatrix} = \begin{bmatrix} \delta x^2 \cdot f_1 - V_0 \\ \delta x^2 \cdot f_2 \\ \delta x^2 \cdot f_3 \\ \vdots \\ \delta x^2 \cdot f_n \\ \vdots \\ \delta x^2 \cdot f_{N-1} \\ \delta x^2 \cdot f_N - V_{N+1} \end{bmatrix} \tag{3.5}$$

which is a simple matrix equation of the form

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b} \tag{3.6}$$

in which $\mathbf{A}$ is tridiagonal. Such equations can be solved by methods which we shall consider in section 3.5. For the moment it should suffice to note that the tridiagonal form can be solved particularly efficiently and that `functions` for this purpose can be found in most libraries of numerical functions.

There are several points which are worthy of note.

- We could only write the equation in this matrix form because the boundary conditions allowed us to eliminate a term from the 1st and last lines.
- Periodic boundary conditions, such as $V(x + L) = V(x)$ can be implemented, but they have the effect of adding a non–zero element to the top right and bottom left of the matrix, A_{1N} and A_{N1}, so that the tridiagonal form is lost.
- It is sometimes more efficient to solve Poisson's or Laplace's equation using Fast Fourier Transformation (FFT). Again there are efficient library routines available (*Numerical Algorithms Group* n.d.). This is especially true in machines with vector processors.

3.4.2 2 or more Dimensions

Poisson's and Laplace's equations can be solved in 2 or more dimensions by simple generalisations of the schemes discussed for 1D. However the resulting matrix will not in general be tridiagonal. The discretised form of the equation takes the form

$$V_{m,n-1} + V_{m,n+1} + V_{m-1,n} + V_{m+1,n} - 4V_{m,n} = \delta x^2 \cdot f_{m,n}. \tag{3.7}$$

The 2 dimensional index pairs $\{m, n\}$ may be mapped on to one dimension for the purpose of setting up the matrix. A common choice is so–called *dictionary order*,

$$(1,1),(1,2),\ldots(1,N),(2,1),(2,2),\ldots(2,N),\ldots(N,1),(N,2),\ldots(N,N)$$

Alternatively Fourier transformation can be used either for all dimensions or to reduce the problem to tridiagonal form.

3.5 Systems of Equations and Matrix Inversion

We now move on to look for solutions of problems of the form

$$\mathbf{AX} = \mathbf{B} \tag{3.8}$$

where $\mathbf{A}$ is an $M \times M$ matrix and $\mathbf{X}$ and $\mathbf{B}$ are $M \times N$ matrices. Matrix inversion is simply the special case where $\mathbf{B}$ is an $M \times M$ unit matrix.

3.5.1 Exact Methods

Most library routines, for example those in the NAG (*Numerical Algorithms Group* n.d.) or LaPack (*Lapack Numerical Library* n.d.) libraries are based on some variation of Gaussian elimination. The standard procedure is to call a first function which performs an $\mathbf{LU}$ decomposition of the matrix $\mathbf{A}$,

$$\mathbf{A} = \mathbf{LU} \tag{3.9}$$

where $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices respectively, followed by a function which performs the operations

$$\mathbf{Y} = \mathbf{L}^{-1}\mathbf{B} \tag{3.10a}$$

$$\mathbf{X} = \mathbf{U}^{-1}\mathbf{Y} \tag{3.10b}$$

on each column of $\mathbf{B}$.

The procedure is sometimes described as *Gaussian Elimination*. A common variation on this procedure is *partial pivoting*. This is a trick to avoid numerical instability in the Gaussian Elimination (or $\mathbf{LU}$ decomposition) by sorting the rows of $\mathbf{A}$ to avoid dividing by small numbers.

There are several aspects of this procedure which should be noted:

- The $\mathbf{LU}$ decomposition is usually done *in place*, so that the matrix $\mathbf{A}$ is overwritten by the matrices $\mathbf{L}$ and $\mathbf{U}$.
- The matrix $\mathbf{B}$ is often overwritten by the solution $\mathbf{X}$.
- A well written $\mathbf{LU}$ decomposition routine should be able to spot when $\mathbf{A}$ is singular and return a flag to tell you so.
- Often the $\mathbf{LU}$ decomposition routine will also return the *determinant* of $\mathbf{A}$.
- Conventionally the lower triangular matrix $\mathbf{L}$ is defined such that all its diagonal elements are unity. This is what makes it possible to replace $\mathbf{A}$ with both $\mathbf{L}$ and $\mathbf{U}$.
- Some libraries provide separate routines for the Gaussian Elimination and the Back–Substitution steps. Often the Back–Substitution must be run separately for each column of $\mathbf{B}$.
- Routines are provided for a wide range of different types of matrices. The symmetry of the matrix is not often used however.

The time taken to solve N equations in N unknowns is generally proportional to N^3 for the Gaussian–Elimination ($\mathbf{LU}$–decomposition) step. The Back–Substitution step goes as N^2 for each column of $\mathbf{B}$. As before this may be modified by parallelism.

3.5.2 Iterative Methods

As an alternative to the above library routines, especially when large *sparse* matrices are involved, it is possible to solve the equations iteratively.

The Jacobi Method

We first divide the matrix $\mathbf{A}$ into 3 parts

$$\mathbf{A} = \mathbf{D} + \mathbf{L} + \mathbf{U} \tag{3.11}$$

where $\mathbf{D}$ is a diagonal matrix (i.e. $D_{ij} = 0$ for $i \neq j$) and $\mathbf{L}$ and $\mathbf{U}$ are *strict* lower and upper triangular matrices respectively (i.e. $L_{ii} = U_{ii} = 0$ for all i). We now write the *Jacobi* or *Gauss–Jacobi* iterative procedure to solve our system of equations as

$$\mathbf{X}^{n+1} = \mathbf{D}^{-1}\left[\mathbf{B} - (\mathbf{L} + \mathbf{U})\,\mathbf{X}^{n}\right] \tag{3.12}$$

where the superscripts ^{n} refer to the iteration number. Note that in practice this procedure requires the storage of the diagonal matrix, $\mathbf{D}$, and a function to multiply the vector, $\mathbf{X}^n$ by $\mathbf{L} + \mathbf{U}$.

This algorithm resembles the iterative solution of hyperbolic (see section 2.3) or parabolic (see section 2.5) partial differential equations, and can be analysed in the same spirit. In particular care must be taken that the method is stable.

Simple C code to implement this for a 1D Poisson's equation is given below.

```
int i;
const int N = ??;  // incomplete code
double xa[N], xb[N], b[N];
while ( ... ) // incomplete code
{
   for ( i = 0; i < N; i++ )
      xa[i] = (b[i] - xb[i-1] - xb[i+1]) * 0.5;
   for ( i = 0; i < N; i++ )
      xb[i] = xa[i];
}
```

Note that 2 arrays are required for `X` and that the matrices, $\mathbf{D}$, $\mathbf{L}$ and $\mathbf{U}$ don't appear explicitely.

The Gauss–Seidel Method

Any implementation of the Jacobi Method (3.12) above will involve a loop over the matrix elements in each column of $\mathbf{X}^{n+1}$. Instead of calculating a completely new matrix $\mathbf{X}^{n+1}$ at each iteration and then replacing $\mathbf{X}^n$ with it before the next iteration, as in the above code, it might seem sensible to replace the elements of $\mathbf{X}^n$ with those of $\mathbf{X}^{n+1}$ as soon as they have been calculated. Naïvely we would expect faster convergence. This is equivalent to rewriting the Jacobi equation (3.12) as

$$\mathbf{X}^{n+1} = (\mathbf{D} + \mathbf{L})^{-1}\left[\mathbf{B} - \mathbf{U}\mathbf{X}^{n}\right]. \tag{3.13}$$

This *Gauss–Seidel* method has better convergence than the Jacobi method, but only marginally so.

As before we consider the example of the solution of the 1D Poisson's equation. As C programs the basic structure might be something like

```
int i;
const int N = ??;  //incomplete code
double x[N], b[N];
while ( ... ) //incomplete code
{
   for ( i = 0; i < N; i++ )
      x[i] = (b[i] - x[i-1] - x[i+1]) * 0.5;
}
```

Note that only one array is now required to store X, whereas the Jacobi method needed 2.

The time required for each iteration is proportional to N for each column of **B**, assuming **A** is sparse. The number of iterations required depends on the details of the problem, on the quality of the initial guess for **X**, and on the accuracy of the required solution.

3.6 Matrix Eigenvalue Problems

No attempt will be made here to describe the detailed algorithms used for matrix diagonalisation. A full discussion can be found in the book by Wilkinson and Reinsch (Wilkinson 1964). The routines found in the common libraries, such as NAG (*Numerical Algorithms Group* n.d.) or LaPack (*Lapack Numerical Library* n.d.) are almost all based on the algorithms in this book.

3.6.1 Schrödinger's equation

In dimensionless form the time–independent Schrödinger equation can be written as

$$-\nabla^2\psi + V(\boldsymbol{r})\psi = E\psi. \tag{3.14}$$

The Laplacian, ∇^2, can be represented in discrete form as in the case of Laplace's or Poisson's (section 3.4) equations. For example, in 1D (3.14) becomes

$$-\psi_{j-1} + (2 + \delta x V_j)\,\psi_j - \psi_{j+1} = E\psi_j \tag{3.15}$$

which can in turn be written in terms of a tridiagonal matrix **H** as

$$\mathbf{H}\underline{\psi} = E\underline{\psi}. \tag{3.16}$$

An alternative and more common procedure is to represent the eigenfunction in terms of a linear combination of *basis functions* so that we have

$$\psi(\boldsymbol{r}) = \sum_\beta a_\beta \phi_\beta(\boldsymbol{r}). \tag{3.17}$$

The basis functions are usually chosen for convenience and as some approximate analytical solution of the problem. Thus in chemistry it is common to choose the ϕ_β to be known atomic orbitals. In solid state physics often plane waves are chosen.

Inserting (3.17) into (3.14) gives

$$\sum_\beta a_\beta \left(-\nabla^2 + V(\boldsymbol{r})\right)\phi_\beta(\boldsymbol{r}) = E\sum_\beta a_\beta \phi_\beta(\boldsymbol{r}). \tag{3.18}$$

Multiplying this by one of the ϕ's and integrating gives

$$\sum_\beta \int \mathrm{d}\boldsymbol{r}\phi^*_\alpha(\boldsymbol{r})\left(-\nabla^2 + V(\boldsymbol{r})\right)\phi_\beta(\boldsymbol{r})a_\beta = E\sum_\beta \int \mathrm{d}\boldsymbol{r}\phi^*_\alpha(\boldsymbol{r})\phi_\beta(\boldsymbol{r})a_\beta. \tag{3.19}$$

We now define 2 matrices

$$\mathbf{H} \equiv H_{\alpha\beta} = \int \mathrm{d}\boldsymbol{r}\phi^*_\alpha(\boldsymbol{r})\left(-\nabla^2 + V(\boldsymbol{r})\right)\phi_\beta(\boldsymbol{r}) \tag{3.20a}$$

$$\mathbf{S} \equiv S_{\alpha\beta} = \int \mathrm{d}\boldsymbol{r}\phi^*_\alpha(\boldsymbol{r})\phi_\beta(\boldsymbol{r}) \tag{3.20b}$$

so that the whole problem can be written concisely as

$$\sum_\beta H_{\alpha\beta}a_\beta = E\sum_\beta S_{\alpha\beta}a_\beta \tag{3.21a}$$

$$\mathbf{H}\boldsymbol{a} = E\mathbf{S}\boldsymbol{a} \tag{3.21b}$$

which has the form of the generalised eigenvalue problem (section 3.6.4). Often the ϕ's are chosen to be orthogonal so that $S_{\alpha\beta} = \delta_{\alpha\beta}$ and the matrix **S** is eliminated from the problem.

3.6.2 General Principles

The usual form of the eigenvalue problem is written

$$\mathbf{A}\boldsymbol{x} = \alpha\boldsymbol{x} \tag{3.22}$$

where $\mathbf{A}$ is a *square* matrix $\boldsymbol{x}$ is an *eigenvector* and α is an *eigenvalue*. Sometimes the eigenvalue and eigenvector are called *latent root* and *latent vector* respectively. An $N \times N$ matrix usually has N distinct eigenvalue/eigenvector pairs[2]. The full solution of the eigenvalue problem can then be written in the form

$$\mathbf{A}\mathbf{U}_r = \mathbf{U}_r\underline{\underline{\alpha}} \tag{3.23a}$$

$$\mathbf{U}_l\mathbf{A} = \underline{\underline{\alpha}}\mathbf{U}_l \tag{3.23b}$$

where $\underline{\underline{\alpha}}$ is a diagonal matrix of eigenvalues and $\mathbf{U}_r$ ($\mathbf{U}_l$) are matrices whose columns (rows) are the corresponding eigenvectors. $\mathbf{U}_l$ and $\mathbf{U}_r$ are the *left* and *right* handed eigenvectors respectively, and $\mathbf{U}_l = \mathbf{U}_r^{-1}$.[3]

- For Hermitian matrices $\mathbf{U}_l$ and $\mathbf{U}_r$ are unitary and are therefore Hermitian transposes of each other: $\mathbf{U}_l^\dagger = \mathbf{U}_r$.
- For Real Symmetric matrices $\mathbf{U}_l$ and $\mathbf{U}_r$ are also real. Real unitary matrices are sometimes called *orthogonal*.

3.6.3 Full Diagonalisation

Routines are available to diagonalise real symmetric, Hermitian, tridiagonal and general matrices.

In the first 2 cases this is usually a 2 step process in which the matrix is first tridiagonalised (transformed to tridiagonal form) and then passed to a routine for diagonalising a tridiagonal matrix. Routines are available which find only the eigenvalues or both eigenvalues and eigenvectors. The former are usually much faster than the latter.

Usually the eigenvalues of a Hermitian matrix are returned sorted into ascending order, but this is not always the case (check the description of the routine). Also the eigenvectors are usually normalised to unity.

For non–Hermitian matrices only the right–handed eigenvectors are returned and are not normalised. In fact it is not always clear what *normalisation* means in the general case.

Some older FORTRAN and all C (not C++) and PASCAL routines for complex matrices store the real and imaginary parts as separate arrays. The eigenvalues and eigenvectors may also be returned in this form. This is due to 2 facts

- The original routines from Wilkinson and Reinsch (Wilkinson 1964) were written in ALGOL, which had no `complex` type.
- Many FORTRAN compilers (even recent ones) handle complex numbers very inefficiently, in that the use a function even for complex addition rather than inline code. In C++ it is worthwhile checking the `complex` header file to see how this is implemented.

3.6.4 The Generalised Eigenvalue Problem

A common generalisation of the simple eigenvalue problem involves 2 matrices

$$\mathbf{A}\boldsymbol{x} = \alpha\mathbf{B}\boldsymbol{x}. \tag{3.24}$$

This can easily be transformed into a simple eigenvalue problem by multiplying both sides by the inverse of either $\mathbf{A}$ or $\mathbf{B}$. This has the disadvantage however that if both matrices are Hermitian $\mathbf{B}^{-1}\mathbf{A}$ is not, and the advantages of the symmetry are lost, together, possibly, with some important physics.

[2] For an exception consider the matrix [2 i / i 0].

[3] Occasionally, $\mathbf{U}_r$ may be singular. In such cases the N vectors do not span an N dimensional space (e.g. 2 are parallel)

There is actually a more efficient way of handling the transformation. Using *Cholesky factorisation* an **LU** decomposition of a positive definite matrix can be carried out such that

$$\mathbf{B} = \mathbf{L}\mathbf{L}^\dagger \tag{3.25}$$

which can be interpreted as a sort of square root of **B**. Using this we can transform the problem into the form

$$\left[\mathbf{L}^{-1}\mathbf{A}\left(\mathbf{L}^\dagger\right)^{-1}\right]\left[\mathbf{L}^\dagger \boldsymbol{x}\right] = \alpha\left[\mathbf{L}^\dagger \boldsymbol{x}\right] \tag{3.26a}$$

$$\mathbf{A}'\boldsymbol{y} = \alpha\boldsymbol{y}. \tag{3.26b}$$

Most libraries contain routines for solving the generalised eigenvalue problem for Hermitian and Real Symmetric matrices using Cholesky Factorisation followed by a standard routine. Problem 6 contains a simple and informative example.

3.6.5 Partial Diagonalisation

Often only a fraction of the eigenvalues are required, sometimes only the largest or smallest. Generally if more than 10% are required there is nothing to be gained by not calculating all of them, as the algorithms for partial diagonalisation are much less efficient *per eigenvalue* than those for full diagonalisation. Routines are available to calculate the M largest or smallest eigenvalues and also all eigenvalues in a particular range.

Sturm Sequence

The Sturm sequence is a very nice algorithm found in most libraries. It finds all the eigenvalues in a given range, $\alpha_{\min} < \alpha < \alpha_{\max}$, and the corresponding eigenvectors. It is also able to find the number of such eigenvalues very quickly and will return a message if insufficient storage has been allocated for the eigenvectors. It does require a tridiagonalisation beforehand and is often combined with the Lanczos (see section 3.6.6) algorithm, to deal with sparse matrices.

3.6.6 Sparse Matrices and the Lanczos Algorithm

None of the above diagonalisation procedures make any use of sparsity. A very useful algorithm for tridiagonalising a sparse matrix is the Lanczos algorithm.

This algorithm was developed into a very useful tool by the Cambridge group of Volker Heine (including (e.g.) Roger Haydock, Mike Kelly, John Pendry) in the late 60's and early 70's.

A suite of programs based on the Lanczos algorithm can be obtained by anonymous ftp from the HENSA[4] archive.

3.7 Problems

1. Poisson's equation $\nabla^2\phi = -\rho/\epsilon_0$ is usually differenced in 2 dimensions as

   $$\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1} - 4\phi_{ij} = S_{ij}$$

   where $S_{ij} = -\delta x^2 \rho_{ij}/\epsilon_0$ and $i, j = 0, 1, 2, \ldots, N-1$.

   The difference equation can be written as a matrix equation $\sum_n A_{mn} f_n = \sigma_m$ where $f_{iN+j} = \phi_{ij}$ and $\sigma_{iN+j} = S_{ij}$.

   Write down the matrix **A** for $N = 3$. Assume the boundaries are at zero potential, i.e. $\phi_{ij} = 0$ iff i or $j = -1$ or i or $j = N$.

[4] ftp://unix.hensa.ac.uk/pub/netlib/lanczos/

2. The equation in question 1 can be solved by the Gauss–Seidel method. If $S_{11} = 1$ and $S_{ij} = 0$ otherwise, find $\phi_{ij}^{(1)}$ and $\phi_{ij}^{(2)}$ if $\phi_{ij}^{(0)} = 0$ for all i and j, where $\phi_{ij}^{(n)}$ is the value of ϕ after n iterations.

3. Work out the factors, L and U, in the LU decomposition of the matrix,

$$\mathbf{A} = \begin{pmatrix} 1 & 3 & 5 \\ 2 & 5 & 3 \\ 1 & 1 & 4 \end{pmatrix}.$$

   Hence,

   - Solve the simultaneous equations,

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b},$$

     for a variety of right hand sides, $\boldsymbol{b}$.
   - Evaluate $\det(\mathbf{A})$
   - Find $\mathbf{A}^{-1}$

4. Show that the Jacobi method (3.12) is stable as long as, μ, the eigenvalue of largest modulus of $\mathbf{D}^{-1}(\mathbf{L} + \mathbf{U})$ is less than unity.

5. Find both eigenvalues and the corresponding left and right handed eigenvectors of the matrix

$$\begin{pmatrix} \epsilon & -1 \\ 1 & 0 \end{pmatrix}.$$

6. The vibrational modes of a certain molecule are described by an equation of motion in the form

$$m_i\omega^2 x_i = K_{ij}x_j,$$

   where m_i and x_i are the mass and the displacement respectively of the ith atom and the real symmetric matrix $\mathbf{K}$ describes the interactions between the atoms.

   Show that this problem can be represented in the form of a generalised eigenvalue problem: $\mathbf{A}\boldsymbol{x} = \lambda\mathbf{B}\boldsymbol{x}$ in which the matrix $\mathbf{B}$ is positive definite.

   By considering the transformation $x_i' = (m_i)^{1/2}x_i$, show how to transform this into a simple eigenvalue problem in which the matrix is real symmetric.

7. Write down the types of matrices which occur in the following problems:

   (a) A simple discretisation (as in question1) of Poisson's equation in 1 dimension.

   (b) The same but in more than 1D.

   (c) A simple discretisation (as above) of Schrödinger's equation in 3 dimensions.

   (d) Schrödinger's equation for a molecule written in terms of atomic basis functions.

   (e) Schrödinger's equation for a crystal at a general $\boldsymbol{k}$ point in the Brillouin zone.

3.8 Project — Oscillations of a Crane

A crane manufacturer wants to know how his newly designed crane is going to behave in a high wind. He wants to know the oscillation frequencies of a steel wire with and without masses on the end. This project is to write a program which calculates the modes of oscillation and their frequencies.

3.8.1 Analysis

The difference equations for the oscillating wire can be derived by dividing it into N segments (10 or 20 should be sufficient) each of which can be considered rigid. This would be the case if the wire consisted of a series of rods connected together.

Let x_n be the displacement of the bottom of the nth segment. Let θ_n be the angle it makes with the vertical. Let T_n be the tension in the nth segment. Assume that the mass of the wire is located at the joints of the segments, Δm at each joint. $\Delta m = \rho \delta z$ where δz is the length of each segment, and ρ is the mass per unit length of the wire. The equation of motion for the mass at the bottom of the nth segment is,

$$\Delta m \frac{\mathrm{d}^2 x_n}{\mathrm{d}t^2} = T_{n+1}\sin(\theta_{n+1}) - T_n \sin(\theta_n) \quad . \tag{3.27}$$

Assume small oscillations so that $\sin(\theta_n) = (x_n - x_{n-1})/\delta z$.

$$\Delta m \frac{\mathrm{d}^2 x_n}{\mathrm{d}t^2} = \frac{1}{\delta z}\left[T_{n+1}x_{n+1} - (T_{n+1} + T_n)x_n + T_n x_{n-1}\right] \quad . \tag{3.28}$$

The mass associated with the end of the wire will be only $\Delta m/2$ since there is no contribution from the (N+1)th segment. Consequently, the equation of motion for this point is,

$$(M + \tfrac{1}{2}\Delta m)\frac{\mathrm{d}^2 x_N}{\mathrm{d}t^2} = -T_N \sin(\theta_N) = -T_N(x_N - x_{N-1})/\delta z \quad , \tag{3.29}$$

where M is any mass carried by the crane. In addition the displacement of the top of the wire is zero, so that $x_0 = 0$ in the equation for $n = 1$.

The modes of oscillation are calculated by seeking solutions of the form $x_n(t) = y_n \exp(i\omega t)$. Substituting this into the equations of motion gives,

$$-\Delta m \omega^2 y_n = \left(T_{n+1}y_{n+1} - (T_n + T_{n+1})y_n + T_n y_{n-1}\right)/\delta z \tag{3.30a}$$

$$-(M + \tfrac{1}{2}\Delta m)\omega^2 y_N = T_N(y_{N-1} - y_N)/\delta z \quad . \tag{3.30b}$$

The specification of the equations is completed by noting that, from the equilibrium conditions,

$$T_N = (M + \tfrac{1}{2}\Delta m)g \quad , \qquad T_n = T_{n+1} + g\Delta m \quad , \tag{3.31}$$

where g is the acceleration due to gravity.

The equations can be organised in the form,

$$\mathbf{A}\boldsymbol{y} = -\mathbf{M}\omega^2 \boldsymbol{y} \quad , \tag{3.32}$$

where $\boldsymbol{y}$ is the column vector of displacements, $(y_1, y_2, \ldots, y_N)$, $\mathbf{A}$ is a symmetric tridiagonal matrix and $\mathbf{M}$ is a diagonal matrix. The problem becomes one of finding the eigenvalues, $-\omega^2$, and eigenvectors, $\boldsymbol{y}$, of a generalised eigenvalue problem. The eigenvectors show how the wire distorts when oscillating in each mode and the eigenvalues give the corresponding oscillation frequencies. Low frequency modes are more important than high frequency modes to the crane manufacturer.

The problem can be solved most easily by using a LaPack routine which finds the eigenvalues and eigenvectors directly. However, before doing so it is necessary to eliminate the matrix $\mathbf{M}$ using the same method as discussed in problem 6.

You should investigate both the computational aspects, such as the dependence of the results on δz, as well as the physical ones, such as the dependence of the behaviour on the mass, M. Do your results make physical sense? You might even compare them with a simple experiment involving a weight on the end of a string.

3.9 Project — Phonons in a Quasicrystal

3.9.1 Introduction

Until a few years ago, it was thought that there were only two different kinds of solids: crystals, in which the atoms are arranged in a regular pattern with translational symmetry (there may be defects, of course);

and amorphous solids, in which there is no long range order, although there is some correlation between the positions of nearby atoms. It was also known that it was impossible for a crystal to have five fold rotational symmetry, since this is incompatible with translational order.

This was how things stood until 1984, when Shechtman et al. (Phys. Rev. Lett. **53** 1951 (1984)), were measuring the X ray diffraction pattern of an alloy of Al and Mn and got a sharp pattern with clear five fold symmetry. The sharpness of the pattern meant that there had to be long range order, but the five fold symmetry meant that the solid could not be crystalline. Shechtman called the material a "quasicrystal".

One possible explanation (although this has still not been conclusively established) is that quasicrystals are three dimensional analogues of Penrose tilings (Scientific American, January 1977 — Penrose tilings were known as a mathematical curiosity before quasicrystals were discovered). Penrose found that you could put together two (or more) different shapes in certain well defined ways so that they "tiled" the plane perfectly, but with a pattern that *never* repeated itself. Sure enough, some of these tilings do have five fold symmetries; and sure enough, there is perfect long range order (although no translational symmetry) so that the diffraction pattern from a Penrose lattice would be sharp.

3.9.2 The Fibonacci Lattice

The mathematical theory of Penrose tilings gets quite high brow and abstruse, but everything is very simple in one dimension. Then the two shapes are lines of different lengths, which we shall call A and C, for Adult and Child (Fibonacci actually studied the dynamics of rabbit populations). Every year each adult has one child and each child becomes an adult. Let us start with a single child

$$C \tag{3.33}$$

and then repeatedly apply the "generation rule,"

$$C \mapsto A \quad , A \mapsto AC \quad , \tag{3.34}$$

to obtain longer and longer sequences. The first few sequences generated are,

$$C \quad A \quad AC \quad ACA \quad ACAAC \quad ACAACACA. \tag{3.35}$$

Note the interesting property that each generation is the "sum" of the 2 previous generations: $ACAAC = ACA \oplus AC$

In a one dimensional Fibonacci quasicrystal, the longs and shorts could represent the interatomic distances; or the strengths of the bonds between the atoms; or which of two different types of atom is at that position in the chain.

3.9.3 The Model

This project is to write a program to work out the phonons (normal mode vibrations) of a Fibonacci quasicrystal with two different sorts of atom. The "adults" and "children" are the masses, M and m, of the two kinds of atom. Since we are only interested in small vibrations, we may represent the forces between the atoms as simple springs (spring constant K) and we will assume that all the springs are identical (K is independent of the types of the atoms at the ends of the spring). The equation of motion of such a system may be written as

$$m_n \frac{\mathrm{d}^2 x_n}{\mathrm{d}t^2} = -m_n \omega^2 x_n = K\left(x_{n+1} - 2x_n + x_{n-1}\right) \quad , \tag{3.36}$$

where x_n and m_n are the displacement and mass of the nth atom. If the chain contains N atoms, you will then have a set of N coupled second order ordinary differential equations.

The choice of boundary conditions for the atoms at either end of the chain is up to you and should not make much difference when the chain is long enough: you could fix the atoms at either end to immovable walls, $x_0 = x_{N+1} = 0$, or you could leave them free by removing the springs at the 2 ends, $K_{0,1} = K_{N,N+1} = 0$. (Periodic boundary conditions — when the chain of atoms is looped around and joined up in a ring — are convenient for analytic work but not so good for numerical work in this case. Why?)

The equations (3.36) are N linear algebraic equations which may be cast as a tridiagonal matrix eigenproblem. The eigenvalues of this problem are ω^2 and so give the vibrational frequencies, and the eigenvectors give the corresponding normal mode coordinates, $x_n, \quad n = 1, N$.

Note, however, that the presence of the masses m_n in (3.36) means that the problem is in the generalised form $\mathbf{A}\boldsymbol{x} = \omega^2\mathbf{B}\boldsymbol{x}$. As discussed in the notes and in problem 6 this can be transformed into the normal eigenvalue problem by making the substitution $x_n = m_n^{-1/2} y_n$ and multiplying the nth equation by $m_n^{-1/2}$. I suggest you solve the eigenproblem by using a NAG or LaPack routine for the eigenvalues only. There should be no problem in dealing with chains of several hundred atoms or more.

3.9.4 The Physics

The idea is to investigate the spectrum (the distribution of the eigenvalues) as the ratio, m/M, of the two masses changes. Your program should list the eigenvalues in ascending order and then plot a graph of eigenvalue against position in the list. When $m = M$, the crystal is "perfect", the graph is smooth, and you should be able to work out all the eigenvalues analytically (easiest when using periodic boundary conditions). But when m and M begin to differ, the graph becomes a a "devil's staircase" with all sorts of fascinating fractal structure. Try to understand the behaviour at small ω (when the wavelength is long and the waves are "acoustic") and the limits as $m/M \to 0$ and $m/M \to \infty$.

Another interesting thing to do is to plot values of m/M (on the y axis) against the vibrational frequencies (on the x axis). Choose a value of m/M, work out all the frequencies, and put a point on the graph for each. The graph now has a line of points parallel to the x axis at the given y value. Do this for a number of values of m/M and see how the spectrum develops as m/M changes. Again, you should try to understand the behaviour when the frequency tends to zero, and the large and small mass ratio limits.

If you have time it is interesting to investigate the fractal structure by focusing in on a single peak for a short sequence and investigating how it splits when you add another "generation". You should find that the behaviour is independent of the number of generations at which you start.

Computational Physics
Monte Carlo Methods and Simulation

4.1 Monte Carlo

The term *Monte–Carlo* refers to a group of methods in which physical or mathematical problems are simulated by using random numbers. Sometimes this is done at a very simple level. For example, calculations of radiation damage in humans have been studied by simulating the firing of random particles into human tissue and randomly carrying out the various possible processes. After a lot of averaging one arrives at the likely damage due to different forms of incident radiation. Similar methods are used to simulate the tracks left in particle physics experiments. Here we will concentrate on 3 different types of calculations using random numbers.

4.1.1 Random Number Generators

Before discussing the uses of random numbers it is useful to have some idea of how random numbers are generated on a computer. Most methods depend on a *chaotic* sequence. The commonest is the *multiplicative congruential method* which relies on prime numbers. Consider the sequence

$$x_{n+1} = (a * x_n) \ \% \ b \tag{4.1}$$

where $x \ \% \ y$ refers to the remainder on dividing x by y and a and b are large integers which have no common factors (often both are prime numbers). This process generates all integers less than b in an apparently random order. After all b integers have been generated the series will repeat itself. Thus one important question to ask about any random number generator is how frequently it repeats itself.

It is worth noting that the sequence is completely deterministic: if the same initial *seed*, x_0 is chosen the same sequence will be generated. This property is extremely useful for debugging purposes, but can be a problem when averaging has to be done over several runs. In such cases it is usual to initialise the *seed* from the system clock.

Routines exist which generate a sequence of integers, as described, or which generate floating point numbers in the range 0 to 1. Most other distributions can be derived from these. There are also very efficient methods for generating a sequence of random bits (Press et al. 1989). The *Numerical Algorithms Group* library contains routines to generate a wide range of distributions.

A difficult but common case is the *Gaussian* distribution. One method simply averages several (say 10) uniform ($0 \rightarrow 1$) random numbers and relies on the central limit theorem. Another method uses the fact that a distribution of complex numbers with both real and imaginary parts Gaussian distributed can also be represented as a distribution of amplitude and phase in which the amplitude has a *Poisson* distribution and the phase is uniformly distributed between 0 and 2π.

As an example of generating another distribution we consider the *Poisson* case,

$$p(y) = \exp(-y). \tag{4.2}$$

Then, if $p'(x)$ and $p(y)$ are the probability distributions of x and y respectively,

$$\int_{-\infty}^{y=f(x)} p(y')\mathrm{d}y' = \int_{-\infty}^{x} p'(x')\mathrm{d}x' \tag{4.3}$$

must be true for all x, as the probability of finding any $y' < y$ must be the same as that for finding any $x' < x$. It follows that

$$p(y) = p'(x)\left|\frac{\mathrm{d}x}{\mathrm{d}y}\right|. \tag{4.4}$$

If x is uniformly distributed between 0 and 1, i.e. $p'(x) = 1$, then

$$p(y) = \frac{\mathrm{d}x}{\mathrm{d}y} = \exp(-y) \qquad \rightarrow \qquad y = -\ln x. \tag{4.5}$$

Hence, a *Poisson* distribution is generated by taking the logarithm of numbers drawn from a uniform distribution.

4.2 Monte–Carlo Integration

Often we are faced with integrals which cannot be done analytically. Especially in the case of multidimensional integrals the simplest methods of discretisation can become prohibitively expensive. For example, the error in a trapezium rule calculation of a d–dimensional integral falls as $N^{-2/d}$, where N is the number of different values of the integrand used. In a Monte–Carlo calculation the error falls as $N^{-1/2}$ independently of the dimension. Hence for $d > 4$ Monte-Carlo integration will usually converge faster.

We consider the expression for the average of a *statistic*, $f(x)$ when x is a random number distributed according to a distribution $p(x)$, then

$$\langle f(x)\rangle = \int p(x)f(x)\mathrm{d}x, \tag{4.6}$$

which is just a generalisation of the well known results for (e.g.) $\langle x\rangle$ or $\langle x^2\rangle$, where we are using the notation $<>$ to denote averaging. Now consider an integral of the sort which might arise while using Laplace transforms.

$$\int_0^\infty \exp(-x)f(x)\mathrm{d}x. \tag{4.7}$$

This integral can be evaluated by generating a set of N random numbers, $\{x_1, \ldots, x_N\}$, from a Poisson distribution, $p(x) = \exp(-x)$, and calculating the mean of $f(x)$ as

$$\frac{1}{N}\sum_{i=1}^{N} f(x_i). \tag{4.8}$$

The error in this mean is evaluated as usual by considering the corresponding *standard error of the mean*

$$\sigma^2 = \frac{1}{N-1}\left(\langle f^2(x)\rangle - \langle f(x)\rangle^2\right). \tag{4.9}$$

As mentioned earlier, Monte–Carlo integration can be particularly efficient in the case of multi–dimensional integrals. However this case is particularly susceptible to the flaws in random number generators. It is a common feature that when a random set of coordinates in a d–dimensional space is generated (i.e. a set of d random numbers), the resulting distribution contains (hyper)planes on which the probability is either significantly higher or lower than expected.

4.3 The Metropolis Algorithm

In statistical mechanics we commonly want to evaluate thermodynamic averages of the form

$$\langle y\rangle = \frac{\sum_i y_i e^{-\beta E_i}}{\sum_i e^{-\beta E_i}} \tag{4.10}$$

where E_i is the energy of the system in state i and $\beta = 1/k_{\mathrm{B}}T$. Such problems can be solved using the Metropolis et al. (1953) algorithm.

Let us suppose the system is initially in a particular state i and we change it to another state j. The detailed balance condition demands that in equilibrium the flow from i to j must be balanced by the flow from j to i. This can be expressed as

$$p_i T_{i\to j} = p_j T_{j\to i} \tag{4.11}$$

where p_i is the probability of finding the system in state i and $T_{i\to j}$ is the probability (or rate) that a system in state i will make a transition to state j. (4.11) can be rearranged to read

$$\frac{T_{i\to j}}{T_{j\to i}} = \frac{p_j}{p_i} \tag{4.12a}$$

$$= e^{-\beta(E_j - E_i)}. \tag{4.12b}$$

Generally the right–hand–side of (4.12) is known and we want to generate a set of states which obey the distribution p_i. This can be achieved by choosing the transition rates such that

$$T_{i\to j} = \begin{cases} 1 & \text{if } p_j > p_i \text{ or } E_j < E_i \\ \dfrac{p_j}{p_i} \text{ or } e^{-\beta(E_j - E_i)} & \text{if } p_j < p_i \text{ or } E_j > E_i \end{cases}. \tag{4.13}$$

In practice if $p_j < p_i$ a random number, r, is chosen between 0 and 1 and the system is moved to state j only if r is less than p_j/p_i or $e^{-\beta(E_j - E_i)}$.

(4.13) is not the only way in which the condition (4.12) can be fulfilled, but it is by far the most commonly used.

An important feature of the procedure is that it is never necessary to evaluate the partition function, the denominator in (4.10) but only the relative probabilities of the different states. This is usually much easier to achieve as it only requires the calculation of the change of energy from one state to another.

Note that, although we have derived the algorithm in the context of thermodynamics, its use is by no means confined to that case. See for example section 4.4.

4.3.1 The Ising model

As a simple example of the Metropolis Method we consider the Ising model of a ferromagnet

$$H = -J \sum_{ij} S_i S_j \tag{4.14}$$

where J is a positive energy, $S = \pm\frac{1}{2}$, and i and j are nearest neighbours on a lattice. In this case we change from one state to another by flipping a single spin and the change in energy is simply

$$\Delta E_i = -J \sum_{j} S_j \tag{4.15}$$

where the sum is only over the nearest neighbours of the flipped spin.

The simulation proceeds by choosing a spin (usually at random) and testing whether the energy would be increased or decreased by flipping the spin. If it is decreased the rules (4.13) say that the spin should definitely be flipped. If, on the other hand, the energy is increased, a uniform random number, r, between 0 and 1 is generated and compared with $e^{-\beta\Delta E}$. If it is smaller the spin is flipped, otherwise the spin is unchanged.

Further information can be found in section 4.7.

4.3.2 Thermodynamic Averages

To average over a thermodynamic quantity it suffices to average over the values for the sequence of states generated by the Metropolis algorithm. However it is usually wise to carry out a number of Monte–Carlo steps before starting to do any averaging. This is to guarantee that the system is in thermodynamic equilibrium while the averaging is carried out.

The sequence of random changes is often considered as a sort of time axis. In practice we think (e.g.) about the time required to reach equilibrium. Sometimes, however, the transition rate becomes very low and the system effectively gets stuck in a non–equilibrium state. This is often the case at low temperatures when almost every change causes an increase in energy.

4.4 Quantum Monte–Carlo

The term *Quantum Monte–Carlo* does not refer to a particular method but rather to any method using Monte–Carlo type methods to solve quantum (usually many–body) problems. As an example consider the evaluation of the energy of a trial wave function $\Psi(\boldsymbol{r})$ where $\boldsymbol{r}$ is a $3N$ dimensional position coordinate of N particles,

$$E = \frac{\int \mathrm{d}\boldsymbol{r}\Psi^*(\boldsymbol{r})\hat{\mathbf{H}}\Psi(\boldsymbol{r})}{\int \mathrm{d}\boldsymbol{r}\Psi^*(\boldsymbol{r})\Psi(\boldsymbol{r})} \tag{4.16}$$

where $\hat{\mathbf{H}}$ is the Hamiltonian operator. This can be turned into an appropriate form for Monte–Carlo integration (see section 4.2) by rewriting as

$$E = \int \mathrm{d}\boldsymbol{r} \left\{ \frac{|\Psi(\boldsymbol{r})|^2}{\int \mathrm{d}\boldsymbol{r}\,|\Psi(\boldsymbol{r})|^2} \right\} \left[\Psi(\boldsymbol{r})^{-1}\hat{\mathbf{H}}\Psi(\boldsymbol{r}) \right] \tag{4.17}$$

such that the quantity in braces ($\{\}$) has the form of a probability distribution. This integral can now easily be evaluated by Monte–Carlo integration. Typically a sequence of $\boldsymbol{r}$'s is generated using the Metropolis (section 4.3) algorithm, so that it is not even necessary to normalise the trial wave function, and the quantity in square brackets [] (4.17) is averaged over the $\boldsymbol{r}$'s.

This method, *variational quantum Monte–Carlo*, presupposes we have a good guess for the wave function and want to evaluate an integral over it. It is only one of several different techniques which are referred to as *quantum Monte Carlo*. Others include, Diffusion Monte–Carlo, Green's function Monte–Carlo and World Line Monte–Carlo.

4.5 Molecular Dynamics

An alternative approach to studying the behaviour of large systems is simply to solve the equations of motion for a reasonably large number of molecules. Usually this is done by using one of the methods described for Ordinary Differential Equations (chapter) to solve the coupled equations of motion for the atoms or molecules to be described. Molecular solids in particular can be studied by considering the molecules as rigid objects and using phenomenological classical equations of motion to describe the interaction between them. As a simple example the *Lennard–Jones* potential

$$\Phi(r) = \epsilon \left(\left(\frac{a}{r}\right)^{12} - 2\left(\frac{a}{r}\right)^{6} \right) \tag{4.18}$$

has been successfully used to describe the thermodynamics of noble gases.

4.5.1 General Principles

Consider a set of particles interacting through a 2–body potential, $\Phi(r)$. The equations of motion can be written in the form

$$\frac{\mathrm{d}\boldsymbol{r}_i}{\mathrm{d}t} = \boldsymbol{v}_i \tag{4.19a}$$

$$\frac{\mathrm{d}\boldsymbol{v}_i}{\mathrm{d}t} = -\frac{1}{m_i}\frac{\partial}{\partial \boldsymbol{r}_i}\sum_{j\neq i}\Phi\left(|\boldsymbol{r}_i - \boldsymbol{r}_j|\right). \tag{4.19b}$$

In practice it is better to use the scalar force $F(r) = \partial\Phi/\partial r$ to avoid unnecessary numerical differentiation.

A common feature of such problems is that the time derivative of one variable only involves the other variable as with $\boldsymbol{r}$ and $\boldsymbol{v}$ in the above equations of motion (4.19). In such circumstances a leap–frog like method suggests itself as the most appropriate. Hence we write

$$\boldsymbol{r}_i^n = \boldsymbol{r}_i^{n-2} + 2\delta t \boldsymbol{v}_i^{n-1} \tag{4.20a}$$

$$\boldsymbol{v}_i^{n+1} = \boldsymbol{v}_i^{n-1} + \frac{2\delta t}{m} \sum_{j \neq i} F\left(|\boldsymbol{r}_i^n - \boldsymbol{r}_j^n|\right) \frac{\boldsymbol{r}_i^n - \boldsymbol{r}_j^n}{|\boldsymbol{r}_i^n - \boldsymbol{r}_j^n|}. \tag{4.20b}$$

This method has the advantage of simplicity as well as the merit of being properly conservative (section 2.6).

The temperature is defined from the kinetic energy of N particles via

$$\tfrac{3}{2} k_{\mathrm{B}} T = \frac{1}{N} \left\langle \sum_i \tfrac{1}{2} m_i \boldsymbol{v}_i^2 \right\rangle \tag{4.21}$$

where the averages $<>$ are taken with respect to time.

As an example of another thermodynamic quantity consider the specific heat at constant volume C_V. This can be calculated by changing the total energy by multiplying all the velocities by a constant amount ($\alpha \approx 1$) and running for some time to determine the temperature. Note that, as the temperature is defined in terms of a time average, multiplying all the velocities by α does not necessarily imply a simple change of temperature, $T \mapsto \alpha^2 T$. At a 1st order phase transition, for example, the system might equilibrate to the same temperature as before with the additional energy contributing to the latent heat.

When the specific heat is known for a range of temperatures it becomes possible, at least in principle, to calculate the entropy from the relationship

$$T \left.\frac{\partial S}{\partial T}\right)_V = \left.\frac{\partial U}{\partial T}\right)_V \tag{4.22}$$

The pressure is rather more tricky as it is defined in terms of the free energy, $F = U - TS$, using

$$p = -\left.\frac{\partial F}{\partial V}\right)_T \tag{4.23}$$

and hence has a contribution from the entropy as well as the internal energy. Nevertheless methods exist for calculating this.

It is also possible to define modified equations of motion which, rather than conserving energy and volume, conserve temperature or pressure. These are useful for describing isothermal or isobaric processes.

Note that for a set of mutually attractive particles it may not be necessary to constrain the volume but for mutually repulsive particles it certainly is necessary.

4.6 Problems

1. Show that (4.1) has the property that it generates **all** the integers from 1 to 10 in an apparently random order if $a = 7$ and $b = 11$, and that the sequence repeats itself thereafter.

2. A certain statistical quantity is distributed according to

   $$p(x) = 2x \quad \text{for } 0 < x < 1$$

   Given a function which generates random numbers uniformly distributed between 0 & 1, show how to transform these into the distribution $p(x)$.

3. Suggest a Monte–Carlo integration procedure for the integral

   $$\int_{-\infty}^{+\infty} e^{-a^2x^2 - b^4x^4} \mathrm{d}x$$

4. Describe how you would use the Metropolis algorithm to generate a set of random numbers distributed according to the integrand of problem 3.

The following are essay type questions which are provided as examples of the sorts of questions which might arise in an examination of Monte–Carlo and related methods.

5. In an ionic conductor, such as AgI, there are several places available on the lattice for each Ag ion, and the ions can move relatively easily between these sites, subject to the Coulomb repulsion between the ions.

   Describe how you would use the Metropolis algorithm to simulate such a system.

6. Some quantum mechanics textbooks suggest that the ground state wave function for a Hydrogen atom is

$$\psi(r) \propto e^{-\alpha r} \quad .$$

   Describe how you would use the variational quantum Monte–Carlo procedure to calculate the ground state energy for a range of values of α and hence how you would provide estimates of the true ground state energy and the value of α.

7. Describe how you would simulate the melting (or sublimation) of Argon *in vacuo* using the molecular dynamics method. Pay particular attention to how you would calculate the specific and latent heats.

4.7 Project — The Ising Model

4.7.1 Introduction

The Ising model for a ferromagnet is not only a very simple model which has a phase transition, but it can also be used to describe phase transitions in a whole range of other physical systems. The model is defined using the equation

$$E = -\sum_{ij} J_{ij} \boldsymbol{S}_i \cdot \boldsymbol{S}_j \tag{4.24}$$

where the i and j designate points on a lattice and $\boldsymbol{S}$ takes the values $\pm\frac{1}{2}$. The various different physical systems differ in the definition and sign of the various J_{ij}'s.

4.7.2 The Model and Method

Here we will consider the simple case of a 2 dimensional square lattice with interactions only between nearest neighbours. In this case

$$E = -J \sum_{i,j_i} \boldsymbol{S}_i \cdot \boldsymbol{S}_{j_i} \tag{4.25}$$

where j_i is only summed over the 4 nearest neighbours of i.

This model can be studied using the Metropolis method as described in the notes, where the state can be changed by flipping a single spin. Note that the change in energy due to flipping the kth spin from $\downarrow$ to $\uparrow$ is given by

$$\Delta E_k = -J \sum_{j_k} \boldsymbol{S}_{j_k}. \tag{4.26}$$

The only quantity which actually occurs in the calculation is

$$Z_k = \exp\left(-\Delta E_k / k_{\mathrm{B}} T\right) \quad , \tag{4.27}$$

and this can only take one of five different values given by the number of neighbouring $\uparrow$ spins. Hence it is sensible to store these in a short array before starting the calculation. Note also that there is really only 1 parameter in the model, $J/k_{\mathrm{B}}T$, so that it would make sense to write your program in terms of this single parameter rather than J and T separately.

The calculation should use periodic boundary conditions, in order to avoid spurious effects due to boundaries. There are several different ways to achieve this. One of the most efficient is to think of the system as a single line of spins wrapped round a torus. This way it is possible to avoid a lot of checking for the boundary. For an $N \times N$ system of spins define an array of $2N^2$ elements using the shortest sensible variable type: `char` in C(++). It is easier to use 1 for spin $\uparrow$ and 0 for spin $\downarrow$, as this makes the calculation of the number of neighbouring $\uparrow$ spins easier. In order to map between spins in a 2d space $S_{\boldsymbol{r}}$ and in the 1d array S_k the following mapping can be used.

$$S_{\boldsymbol{r}+\delta\boldsymbol{x}} \mapsto S_{k+1}, \quad S_{\boldsymbol{r}-\delta\boldsymbol{x}} \mapsto S_{k+N^2-1}, \quad S_{\boldsymbol{r}+\delta\boldsymbol{y}} \mapsto S_{k+N}, \quad S_{\boldsymbol{r}-\delta\boldsymbol{y}} \mapsto S_{k+N^2-N} \tag{4.28}$$

where the 2nd N^2 elements of the array are always maintained equal to the 1st N^2. This way it is never necessary to check whether one of the neighbours is over the edge. It is important to remember to change S_{k+N^2} whenever S_k is changed.

The calculation proceeds as follows:

1. Initialise the spins, either randomly or aligned.
2. Choose a spin to flip. It is better to choose a spin at random rather than systematically as systematic choices can lead to spurious temperature gradients across the system.
3. Decide whether to flip the spin by using the Metropolis condition (see notes).
4. If the spin is to be flipped, do so but remember to flip its mirror in the array.
5. Update the energy and magnetisation.
6. Add the contributions to the required averages.
7. Return to step 2 and repeat.

4.7.3 The Physics

In general it is advisable to run the program for some time to allow it to reach equilibrium before trying to calculate any averages. Close to a phase transition it is often necessary to run for much longer to reach equilibrium. The behaviour of the total energy during the run is usually a good guide to whether equilibrium has been reached. The total energy, E, and the magnetisation can be calculated from (4.24) and

$$M = \frac{1}{N}\sum_i S_i \tag{4.29}$$

It should be possible to calculate these as you go along, by accumulating the changes rather than by recalculating the complete sum after each step. A 10×10 lattice should suffice for most purposes and certainly for testing, but you may require a much bigger lattice close to a transition.

A useful trick is to use the final state at one temperature as the initial state for the next slightly different temperature. That way the system won't need so long to reach equilibrium.

It should be possible to calculate the specific heat and the magnetic susceptibility. The specific heat could be calculated by differentiating the energy with respect to temperature. This is a numerically questionable procedure however. Much better is to use the relationship

$$C_v \propto \frac{1}{N}\left(\frac{J}{k_{\mathrm{B}}T}\right)^2 \left(\langle E^2\rangle - \langle E\rangle^2\right) \tag{4.30}$$

Similarly, in the paramagnetic state, the susceptibility can be calculated using

$$\chi \propto \frac{1}{N}\frac{J}{k_{\mathrm{B}}T}\left(\langle S^2\rangle - \langle S\rangle^2\right) \tag{4.31}$$

where $S = \sum_i S_i$ and the averages are over different states, i.e. can be calculated by averaging over the different Metropolis steps. Both these quantities are expected to diverge at the transition, but the divergence

will tend to be rounded off due to the small size of the system. Note however that the fact that (4.30) & (4.31) have the form of variances, and that these diverge at the transition, indicates that the average energy and magnetisation will be subject to large fluctuations around the transition.

Finally a warning. A common error made in such calculations is to add a contribution to the averages only when a spin is flipped. In fact this is wrong as the fact that it isn't flipped means that the original state has a higher probability of occupation.

4.8 Project — Quantum Monte Carlo Calculation

4.8.1 Introduction

This project is to use the variational quantum Monte Carlo method to calculate the ground state energy of the He atom. The He atom is a two electron problem which cannot be solved analytically and so numerical methods are necessary. Quantum Monte Carlo is one of the more interesting of the possible approaches (although there are better methods for this particular problem).

4.8.2 The Method

The Schrödinger equation for the He atom in atomic units is,

$$\left(-\frac{1}{2}\nabla_{r_1}^2 - \frac{1}{2}\nabla_{r_2}^2 - \frac{2}{r_1} - \frac{2}{r_2} + \frac{1}{r_{12}}\right)\Psi(\boldsymbol{r}_1,\boldsymbol{r}_2) = E\Psi(\boldsymbol{r}_1,\boldsymbol{r}_2) \quad , \tag{4.32}$$

where $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are the position vectors of the two electrons, $r_1 = |\boldsymbol{r}_1|$, $r_2 = |\boldsymbol{r}_2|$, and $r_{12} = |\boldsymbol{r}_1 - \boldsymbol{r}_2|$. Energies are in units of the Hartree energy (1 Hartree = 2 Rydbergs) and distances are in units of the Bohr radius. The ground state spatial wavefunction is symmetric under exchange of the two electrons (the required antisymmetry is taken care of by the spin part of the wavefunction, which we can forget about otherwise).

The expression for the energy expectation value of a particular trial wavefunction, $\Phi_T(\boldsymbol{r}_1,\boldsymbol{r}_2)$, is,

$$E_T = \frac{\int\ldots\int \Phi_T^* \hat{H} \Phi_T \mathrm{d}^3r_1 \mathrm{d}^3r_2}{\int\ldots\int \Phi_T^* \Phi_T \mathrm{d}^3r_1 \mathrm{d}^3r_2} \quad . \tag{4.33}$$

In the variational Monte Carlo method, this equation is rewritten in the form,

$$E_T = \int\ldots\int \left(\frac{1}{\Phi_T}\hat{H}\Phi_T\right) f\left(\boldsymbol{r}_1,\boldsymbol{r}_2\right) \mathrm{d}^3r_1 \mathrm{d}^3r_2 \quad , \tag{4.34}$$

where

$$f(\boldsymbol{r}_1,\boldsymbol{r}_2) = \frac{\Phi_T^*(\boldsymbol{r}_1,\boldsymbol{r}_2)\Phi_T(\boldsymbol{r}_1,\boldsymbol{r}_2)}{\int\ldots\int \Phi_T^* \Phi_T \mathrm{d}^3r_1 \mathrm{d}^3r_2} \tag{4.35}$$

is interpreted as a probability density which is sampled using the Metropolis algorithm. Note that the Metropolis algorithm only needs to know *ratios* of the probability density at different points, and so the normalisation integral,

$$\int\ldots\int \Phi_T^* \Phi_T \mathrm{d}^3r_1 \mathrm{d}^3r_2 \quad , \tag{4.36}$$

always cancels out and does not need to be evaluated.

The mean of the values of the "local energy",

$$\frac{1}{\Phi_T}\hat{H}\Phi_T \quad , \tag{4.37}$$

at the various points along the Monte Carlo random walk then gives an estimate of the energy expectation value. By the variational principle, the exact energy expectation value is always greater than or equal to the true ground state energy; but the Monte Carlo estimate has statistical errors and may lie below the true

ground state energy if these are large enough. Anyway, the better the trial wavefunction, the closer to the true ground state energy the variational estimate should be.

In this project you are given a possible trial wavefunction,

$$\Phi = e^{-2r_1} e^{-2r_2} e^{r_{12}/2} \quad . \tag{4.38}$$

Use the variational Monte Carlo technique to calculate variational estimates of the true ground state energy.

Before you start programming, you will need the analytic expressions for the local energy. This involves some nasty algebra, but the answer is,

$$\frac{1}{\Phi}\hat{H}\Phi = -\frac{17}{4} - \frac{\boldsymbol{r}_1 \cdot \boldsymbol{r}_{12}}{r_1 r_{12}} + \frac{\boldsymbol{r}_2 \cdot \mathbf{r}_{12}}{r_2 r_{12}} \quad . \tag{4.39}$$

where $\boldsymbol{r}_1$ and $\boldsymbol{r}_2$ are the coordinates of the 2 atoms relative to the nucleus and $\boldsymbol{r}_{12} = \boldsymbol{r}_2 - \boldsymbol{r}_1$.

The Monte Carlo moves can be made by generating random numbers (use a library routine to do this) and adding them to the electron coordinates. I suggest that you update all six electron position coordinates $(x_1, y_1, z_1, x_2, y_2, z_2)$ each move, and so you will need six random numbers each time. The accepted lore is that the Metropolis algorithm is most efficient when the step size is chosen to keep the acceptance probability close to 0.5. However, the method should work in principle no matter what the step size and you should try a few different step sizes to confirm that this is indeed the case. The starting positions of the two electrons can be chosen randomly, but remember that the Metropolis algorithm only samples the probability distribution exactly in the limit as the number of moves tends to infinity. You will therefore have to throw away the results from the moves near the beginning of the run and only start accumulating the values of the local energy once things have settled down. You should experiment to find out how many moves you need to throw away.

The statistical errors in Monte Carlo calculations decrease like $1/\sqrt{N}$, where N is the total number of moves after the initial equilibration period. The errors therefore improve only slowly as the length of the run is increased. You will not be able (and should not attempt) to attain great accuracy. However, you should think hard about the magnitude of the statistical errors involved. Calculating the variance of the values in the list of energies accumulated during the random walk is easy and you should certainly do it.

4.8.3 The Physics

What are your best estimates of the ground state energy and the corresponding statistical error? Can you see any physics behind the form of the trial wavefunction?

Computational Physics
Computer Algebra

5.1 Introduction

Algebraic or Symbolic computing is very different from the sort of numerical computing you have learned so far. Algebraic computing systems have been available for a long time, almost as long as the older conventional computer languages such as FORTRAN or ALGOL. Systems such as REDUCE or MACSYMA were available on mainframes in the 1960's as general purpose packages. There were also some specialist packages available for various branches of physics and mathematics, such as the program FORM which is widely used in theoretical particle physics (and is available free). With the development of PCs packages such as muMATH became available and more recently MAPLE and MATHEMATICA. In these lectures the examples will be drawn from MATHEMATICA but the discussion will be kept as general as possible.

The newer packages not only perform algebraic manipulations but have powerful built in numerical analysis and plotting capabilities. Most systems can be used as *interpreters,* with the commands typed in one by one with an instant (?) response, but many also include the possibility to define and run your own macros or procedures, so that they have many of the attributes of a conventional compiler as well.

As many of these packages were designed with different specialised criteria in mind, some packages are better for particular classes of problems than others.

5.2 Basic Principles

In this section we consider some of the basic commands of Mathematica(Wolfram 1991) and how they can be used. We will make no attempt at completeness. Any work done using computer algebra will require a copy of the Mathematica handbook(Wolfram 1991).

Note firstly the general point that the system is *Case Sensitive*: `pi` and `Pi` are different (the latter representing π). Different sorts of brackets are used for different purposes

`( )`	for grouping	$(a+b)(a-b)$
`[ ]`	for functions	$f[x]$ or $Sin[x]$
`{}`	for lists	see examples later
`[ [ ] ]`	for indexing	$a[[n]] \equiv a_n$

In the examples to be given below prompts by the computer are shown in *italics*, things typed by the user in `typewriter font` and responses from the computer in normal type.

Variables are manipulated as in algebra unless they have been given a particular value:

In[1]:= `y = (1 + x)^2`
Out[1]= $(1+x)^2$
In[2]:= `x = 2`
Out[2]= 2
In[3]:= `y`
Out[3]= 9

As well as being given a permanent value variables can be given a temporary value

```
In[4]:=   Clear[x]
In[5]:=   y = (1 + x)^2
Out[5]=   (1 + x)^2
In[6]:=   y/.  x -> 2
Out[6]=   9
In[7]:=   y
Out[7]=   (1 + x)^2
```

Note the command `Clear[x]` which makes sure that x does not have a value.
Basic algebra can be done by using the functions `Expand[]` and `Factor[]` as follows

```
In[8]:=   Expand[y]
Out[8]=   1 + 2x + x^2
In[9]:=   Factor[%]
Out[9]=   (1 + x)^2
```

Note the use of `%` as shorthand for the result of the last operation. We could also have written `Factor[Out[8]]` here.

There are several other commands available which do specific manipulations on such expressions, in particular:

- `Simplify[]` tries to find the simplest form of an expression.
- `Together[]` put all terms over a common denominator.
- `Collect[expr,x]` groups together powers of x in the expression.
- `Coefficient[expr,x^4]` picks out the coefficient of `x^4` in the expression.

Differentiation and integration are carried out using the `D[f,x]` and `Integrate[f,x]` operators.

```
In[10]:=   D[y,x]
Out[10]=   2(1 + x)
In[11]:=   Integrate[%,x]
Out[11]=   2x + x^2
In[12]:=   Factor[%]
Out[12]=   x(2 + x)
In[13]:=   Integrate[y,x]
Out[13]=   x + x^2 + x^3/3
```

or, using special functions,

```
In[14]:=   D[Sin[x],x]
Out[14]=   Cos[x]
In[15]:=   Integrate[%,x]
Out[15]=   Sin[x]
In[16]:=   x = Sin[z]
Out[16]=   Sin[z]
In[17]:=   y
Out[17]=   (1 + Sin[z])^2
In[18]:=   D[y,z]
Out[18]=   2 Cos[z](1 + Sin[z])
In[19]:=   Integrate[y,z]
Out[19]=   3z/2 - 2 Cos[z] - Sin[2z]/4
```

Higher derivatives are written as

- `D[f, x, y, z, ...]` multiple derivative $\frac{\partial}{\partial x}\frac{\partial}{\partial y}\frac{\partial}{\partial z}\ldots$.
- `D[f, {x,n}]` repeated derivative $\frac{\partial^n}{\partial x^n}$.

whereas definite integrals have the format

- `Integrate[f, {x,a,b}]` definite integral $\int_a^b \mathrm{d}x$.
- `Integrate[f, {x,0,Infinity}]` definite integral $\int_0^\infty \mathrm{d}x$.

Whereas differentiation always has a well–defined answer, sometimes *Mathematica* is unable to find an analytical result. In such cases the function `N[]` can be used to obtain a numerical result.

```
In[20]:= Integrate[Sin[Sin[x]], {x,0,1}]
Out[20]= Integrate[Sin[Sin[x]], {x, 0, 1}]
In[21]:= N[%]
Out[21]= 0.430606
In[22]:= N[%%,40]
Out[22]= 0.4306061031206906049123773552484 66
```

where the last command evaluates the 2nd last (`%%`) result to `40` significant figures. This function can also be used in other situations such as

```
In[23]:= N[Pi, 100]
Out[23]= 3.14159265358979323846264338327950288419716939937 51
         05820974944592307816406286208998628034825342117068
```

Mathematica has a number of built in constants (all beginning with capital letters) such as

```
In[24]:= E^(I Pi)
Out[24]= -1
```

5.3 Solving Equations

Before considering how to solve simple equations using Mathematica we have to note one other detail of notation, == is the symbol for Boolean equality. So, for example

```
In[25]:= x=4
Out[25]= 4
In[26]:= x==4
Out[26]= True
In[27]:= x==5
Out[27]= False
In[28]:= x
Out[28]= 4
```

from which we see that the result of `x==y` is either "True" or "False". Contrast this with the symbol = which is used to assign a value to a symbol.

Now we introduce the `Solve` function for solving equations. the syntax can be seen from the following examples:

```
In[29]:= Solve[4x + 8 == 0, x]
Out[29]= {{x -> -2}}
In[30]:= Solve[x^2 + 2x - 7 == 0, x]
Out[30]= {{x -> (-2 - 4 Sqrt[2])/2}, {x -> (-2 + 4 Sqrt[2])/2}}
In[31]:= N[%]
Out[31]= {{x -> -3.82843}, {x -> 1.82843}}
In[32]:= Solve[a x^2 + b x + c == 0, x]
Out[32]= {{x -> (-b + Sqrt[b^2 - 4ac])/(2a)}, {x -> (-b - Sqrt[b^2 - 4ac])/(2a)}}
```

where both numerical and *symbolic* solutions may be obtained. We can also find more complicated problems, such as

```
In[33]:= Solve[{x^2 + y^2 == 1, x + 3 y == 0}, {x, y}]
Out[33]= {{x -> -3/Sqrt[10], y -> 1/Sqrt[10]}, {x -> 3/Sqrt[10], y -> -(1/Sqrt[10])}}
In[34]:= x/.%
Out[34]= {x -> -3/Sqrt[10]}, {x -> 3/Sqrt[10]}
In[35]:= y/.%%
Out[35]= {y -> 1/Sqrt[10]}, {y -> -(1/Sqrt[10])}
```

in which we should note the syntax for solving systems of equations by grouping things in braces, {}, and for extracting parts of the solution. Note also how the different, distinct solutions are grouped together.

Unfortunately there are equations which Mathematica cannot solve so simply, but which with a little help it could do so. One very useful command in this context is `Eliminate`, used as follows

```
In[36]:= Eliminate[{a x + y == 0, 2 x + (1 - a) y == 1}, y]
Out[36]= -(ax) + a^2 x == 1 - 2x
In[37]:= Solve[%, x]
Out[37]= {{x -> 1/(2 - a + a^2)}}
```

Often Mathematica is unable to solve a problem in a single step but can do so if the user *guides* it through the steps. This is particularly the case when an equation may have unphysical solutions which can be ignored. The user can recognise the physically important solutions and concentrate on those.

5.4 Differential Equations

The function `DSolve` is used to solve differential equations. Note the following examples

```
In[38]:= DSolve[y'[x] == a y[x], y[x], x]
Out[38]= {{y[x] -> E^(ax) C[1]}}
In[39]:= DSolve[{y'[x] == a y[x], y[0] == 1}, y[x], x]
Out[39]= {{y[x] -> E^(ax)}}
```

where we can see several important features. The notation `y'[x]` has the obvious meaning of 1st derivative of the function `y[x]` with respect to `x`. The 2nd and 3rd parameters of the `DSolve` function are the unknown function and the variable of integration respectively. Note also the appearance of the arbitrary constant `C[1]` in the 1st example and the inclusion of a boundary condition `y[0] == 1` in the 2nd.

Mathematica can handle more general systems of differential equations, but solutions are usually only found if the equations are linear, unless the user is able to fill in some intermediate steps.

There are also functions `NSolve` and `NDSolve` which provide numerical solutions when the analytical solutions are not available from `Solve` or `DSolve`.

5.5 Other Features

We do not have enough time here to list all the powerful facilities within Mathematica, but we present a short list of some of the more useful ones.

- Functions to expand, sum, and take the limit of series and functions: `Series, Sum, Limit`.
- Plotting and Fitting numerical data and solutions of (e.g.) differential equations.
- Facilities to manipulate vectors and matrices in different coordinate systems.
- Statistical functions.
- A programming language using which it is possible to write your own symbolic programs and hence to expand Mathematica's facilities.
- A library of prewritten extensions for various specialist purposes as well as some specially written code available from other sources.

5.6 Final Example

Computer Algebra or Symbolic Computing is particularly useful when a problem involves a large amount of very tedious work where it is easy to miss or lose important terms. One such is the solution of non–linear differential equations using a series expansion. We therefore illustrate such a solution where we are trying to solve the equation

$$\frac{\mathrm{d}y}{\mathrm{d}x} = ay^2. \tag{5.1}$$

In[40]:= `y = a0 + a1 x + a2 x^2 + a3 x^3 + a4 x^4`

Out[40]= $a0 + a1\,x + a2\,x^2 + a3\,x^3 + a4\,x^4$

In[41]:= `D[y, x] - a (y)^2`

Out[41]= $a1 + 2\,a2\,x + 3\,a3\,x^2 + 4\,a4\,x^3$
$-a(a0 + a1\,x + a2\,x^2 + a3\,x^3 + a4\,x^4)^2$

In[42]:= `equ = Expand[%]`

Out[42]= $-(a\,a0^2) + a1 - 2\,a\,a0\,a1\,x + 2\,a2\,x - a\,a1^2\,x^2 - 2\,a\,a0\,a2\,x^2$
$+3\,a3\,x^2 - 2\,a\,a1\,a2\,x^3 - 2\,a\,a0\,a3\,x^3 + 4\,a4\,x^3$
$-a\,a2^2\,x^4 - 2\,a\,a1\,a3\,x^4 - 2\,a\,a0\,a4\,x^4 - 2\,a\,a2\,a3\,x^5$
$-2\,a\,a1\,a4\,x^5 - a\,a3^2\,x^6 - 2\,a\,a2\,a4\,x^6 - 2\,a\,a3\,a4\,x^7$
$-a\,a4^2\,x^8$

In[43]:= `equ/.x->0`

Out[43]= $-(a\,a0)^2 + a1$

In[44]:= `Solve[%==0, a1]`

Out[44]= $\{\{a1 \to a\,a0^2\}\}$

In[45]:= `a1 = a1/.%[[1]]`

Out[45]= $a\,a0^2$

In[46]:= `equ`

Out[46]= $-2\,a^2\,a0^3\,x + 2\,a2\,x - a^3\,a0^4\,x^2 - 2\,a\,a0\,a2\,x^2 + 3\,a3\,x^2$
$-2\,a^2\,a0^2\,a2\,x^3 - 2\,a\,a0\,a3\,x^3 + 4\,a4\,x^3 - a\,a2^2\,x^4$
$-2\,a^2\,a0^2\,a3\,x^4 - 2\,a\,a0\,a4\,x^4 - 2\,a\,a2\,a3\,x^5 - 2\,a^2\,a0^2\,a4\,x^5$
$-a\,a3^2\,x^6 - 2\,a\,a2\,a4\,x^6 - 2\,a\,a3\,a4\,x^7 - a\,a4^2\,x^8$

In[47]:= `Coefficient[equ, x]`

Out[47]= $-2\,a^2\,a0^3 + 2\,a2$

In[48]:= `Solve[%==0, a2]`

Out[48]= $\{\{a2 \to a^2\,a0^3\}\}$

In[49]:= `a2 = a2/.%[[1]]`

Out[49]= $a^2\,a0^3$

```
In[50]:= equ
Out[50]= -3 a^3 a0^4 x^2 + 3 a3 x^2 - 2 a^4 a0^5 x^3 - 2 a a0 a3 x^3
         +4 a4 x^3 - a^5 a0^6 x^4 - 2 a^2 a0^2 a3 x^4 - 2 a a0 a4 x^4
         -2 a^3 a0^3 a3 x^5 - 2 a^2 a0^2 a4 x^5 - a a3^2 x^6 - 2 a^3 a0^3 a4 x^6
         -2 a a3 a4 x^7 - a a4^2 x^8
In[51]:= Coefficient[equ, x^2]
Out[51]= -3 a^3 a0^4 + 3 a3
In[52]:= Solve[%==0, a3]
Out[52]= {{a3 -> a^3 a0^4}}
In[53]:= a3 = a3/.%[[1]]
Out[53]= a^3 a0^4
```

```
In[54]:= equ
Out[54]= -4 a^4 a0^5 x^3 + 4 a4 x^3 - 3 a^5 a0^6 x^4 - 2 a a0 a4 x^4
         -2 a^6 a0^7 x^5 - 2 a^2 a0^2 a4 x^5 - a^7 a0^8 x^6 - 2 a^3 a0^3 a4 x^6
         -2 a^4 a0^4 a4 x^7 - a a4^2 x^8
In[55]:= Coefficient[equ, x^3]
Out[55]= -4 a^4 a0^5 + 4 a4
In[56]:= Solve[%==0, a4]
Out[56]= {{a4 -> a^4 a0^5}}
In[57]:= a4 = a4/.%[[1]]
Out[57]= a^4 a0^5
In[58]:= y
Out[58]= a0 + a a0^2 x + a^2 a0^3 x^2 + a^3 a0^4 x^3 + a^4 a0^5 x^4
In[59]:= Factor[%]
Out[59]= a0 (1 + a a0 x + a^2 a0^2 x^2 + a^3 a0^3 x^3 + a^4 a0^4 x^4)
```

This looks like the first few terms of the series for

$$y(x) = \frac{a_0}{1 - a_0 a x} \tag{5.2}$$

which is indeed the solution of the differential equation (5.1. So we see that in this sort of *bookkeeping* exercise the use of computer algebra can be of considerable assistance.

5.7 Project — The Thomas–Fermi Approximation

5.7.1 Introduction

Consider a system of free electrons or of electrons in a semiconductor within the effective mass approximation. Since the density of states for free electrons, $\rho(E) \propto E^{1/2}$, the density of electrons, n, below a chemical potential μ is $n \propto \mu^{3/2}$. If an electrical potential energy, V is added while keeping the electrochemical potential energy (or Fermi level), $E_F = \mu + V$, constant, then the electron density changes to $n \propto (E_F - V)^{3/2}$.

In the Thomas–Fermi approximation this charge density is assumed to be a local quantity. Poisson's equation for the electrical potential then takes the form

$$\frac{\mathrm{d}^2 V}{\mathrm{d}x^2} \propto (E_F - V)^{3/2} \quad , \tag{5.1}$$

which can be simplified into the form

$$\frac{\mathrm{d}^2 \phi}{\mathrm{d}x^2} = A\phi^{3/2} \quad . \tag{5.2}$$

This is a second order differential equation, so the general solution must contain 2 unknowns. One solution is known, namely

$$\phi = \frac{400}{A^2(x - x_0)^4} \quad , \tag{5.3}$$

which contains the single unknown x_0.

The object of the exercise here is to find a more general solution, or at least something which may be used as such (i.e. a series expansion).

5.7.2 Some Ideas

One possible approach would be to use the Frobenius method by assuming

$$\phi(x) = x^s \sum_{n=0}^{\infty} a_n x^n \quad , \tag{5.4}$$

but this presents 2 major difficulties.

- It doesn't adequately take account of the known solution, (5.3).
- The $3/2$ power cannot be easily expanded.

For this reason I suggest an alternative approach: make the substitution $\phi(x) = y^2(x)$ so that (5.2) then takes the form

$$\frac{\mathrm{d}^2 y^2}{\mathrm{d}x^2} = Ay^3 \quad , \tag{5.5}$$

and then use the Frobenius expansion in $z = x - x_0$ to solve the equation. Thus

$$y(z) = z^s \sum_{n=0}^{\infty} b_n z^n \quad . \tag{5.6}$$

By considering the limit of small z and the known solution (5.3) we see immediately that $s = -2$ and that $b_0 = 20/A$.

You should try to find some more coefficients in the expansion (5.6) using *Mathematica*. The technique is similar to that described in the notes. You should bear in mind that we have already found one arbitrary coefficient, x_0, so that your solution should contain at least one other such arbitrary coefficient. Some of the coefficients may be zero; you should try to find a few non–trivial ones.

In your report you should describe how you have obtained the solution and any special insight you have gained into the nature of the solution. Instead of a program as an appendix you might consider including a printout of a Mathematica *Notebook* as well as some graphs of your solution.

Bibliography

Lapack Numerical Library n.d.
*ftp://unix.hensa.ac.uk/pub/netlib/lapack/

Metropolis N, Rosenbluth A W, Rosenbluth M N, Teller A H & Teller E 1953 *J. Chem. Phys.* **21**, 1087.

Numerical Algorithms Group n.d.
*http://www.nag.co.uk:70/

Potter D 1973 *Computational Physics* Wiley Chichester.

Press W H, Flannery B P, Teukolsky S A & Vettering W T 1989 *Numerical Recipes: The Art of Scientific Computing* Cambridge University Press Cambridge.

Wilkinson J H 1964 *The Algebraic Eigenvalue Problem* Clarendon Press Oxford.

Wolfram S 1991 *Mathematica — A System for Doing Mathematics by Computer* Addison-Wesley Redwood City, California. ISBN 0 201 51502 4.